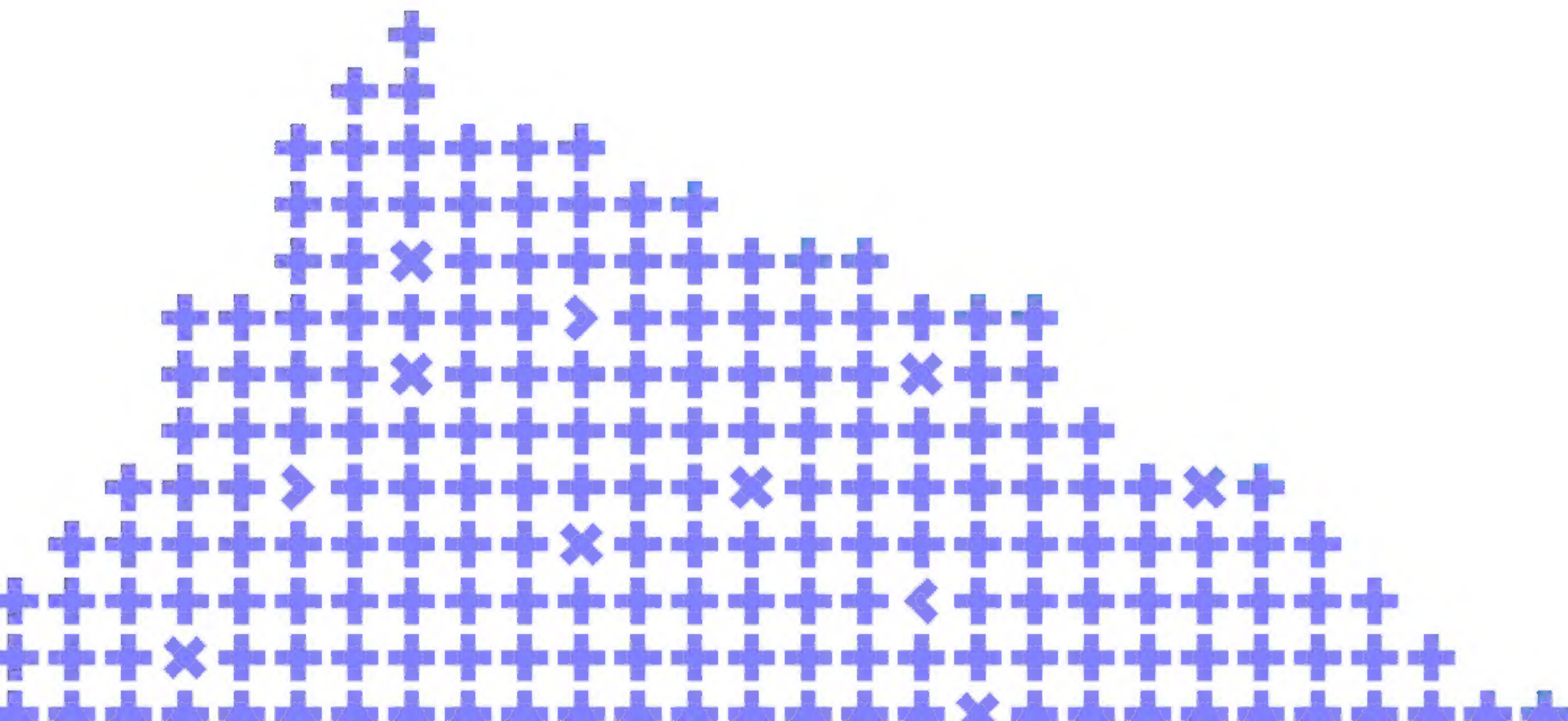


How we reduced logs costs by moving from Elasticsearch to Grafana Loki

Igor Latkin



Co-organizer

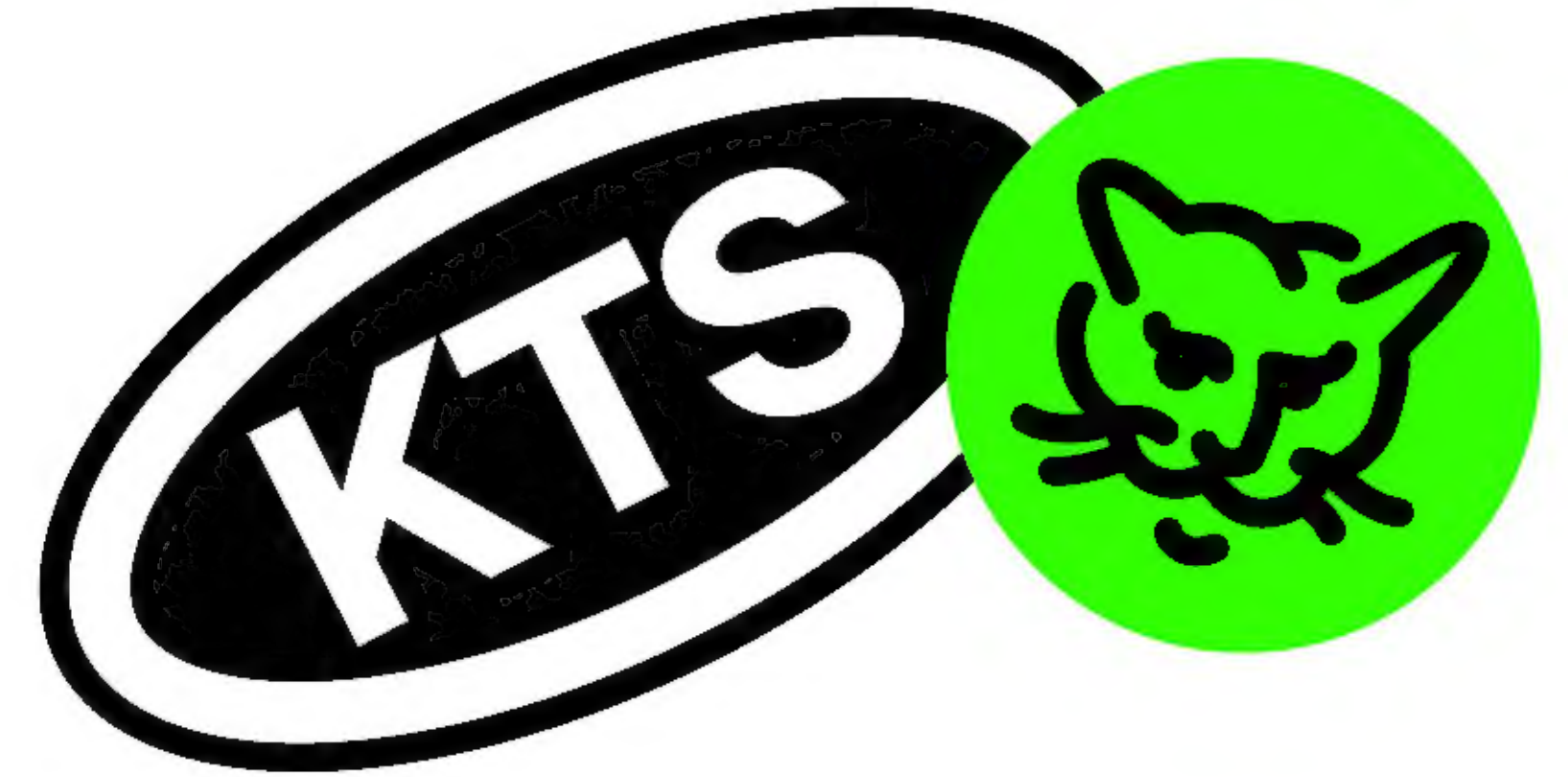
Yandex

WHO AM I?

IGOR LATKIN

Co-founder & System Architect @ KTS

- Corporate systems
- Non-standard projects
- Mobile
- DevOps





AGENDA

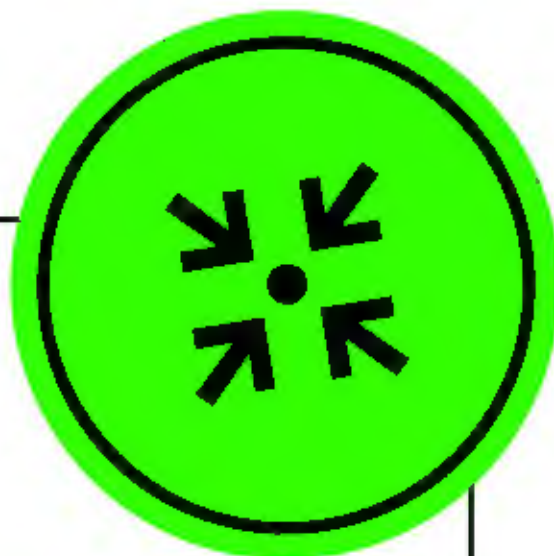
AGENDA

1. Log collection task
2. Loki architecture
3. Our journey of logs transferring from ES to Loki
4. (bonus) Loki configuration tips & tricks

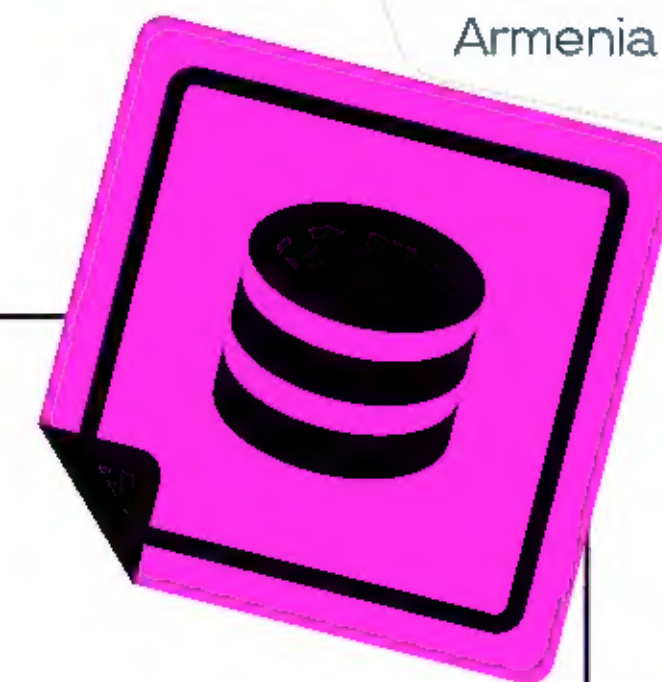
WHAT THIS TALK IS NOT

- Loki or Elasticsearch tutorial
- Loki vs Elasticsearch comprehensive comparison
- Complete set of instructions how to transfer logs in your environment

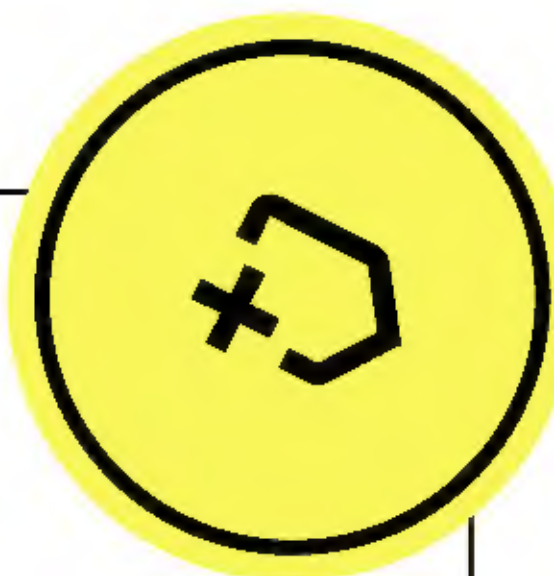
LOG COLLECTION PROBLEM



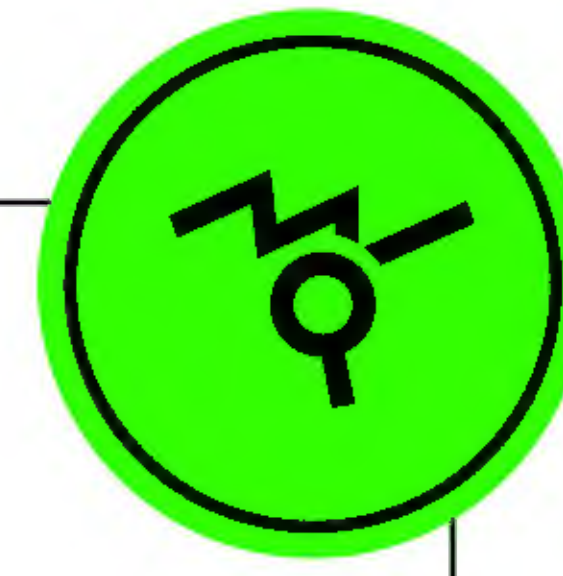
How to collect



How to store



How to extract
metadata

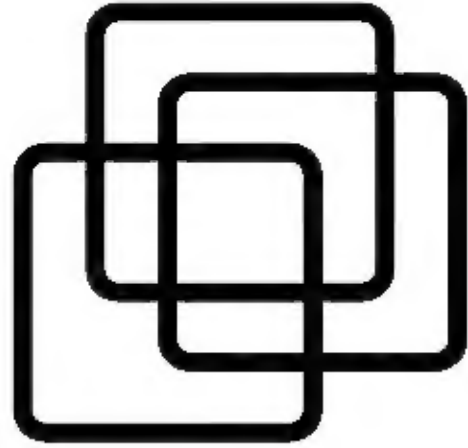


How to query

MULTIPLE SOURCES



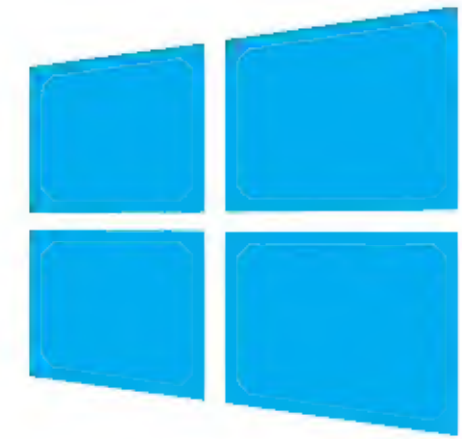
KUBERNETES CLUSTER



VIRTUAL MACHINES



CLOUD ENVS



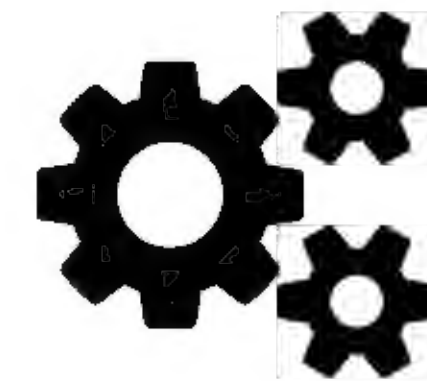
WINDOWS EVENTS



DATABASES



TRAFFIC SNIFFING

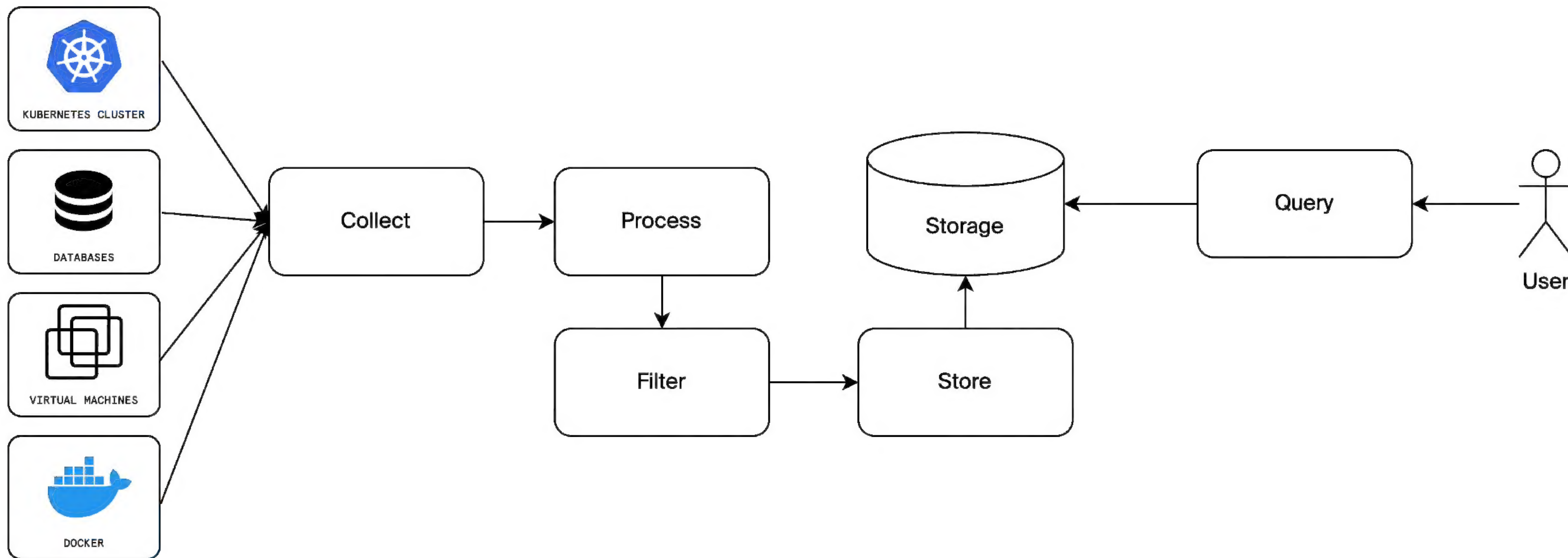


SYSLOG



DOCKER

LOGGING PIPELINE



WE ALL KNOW THEM



elastic

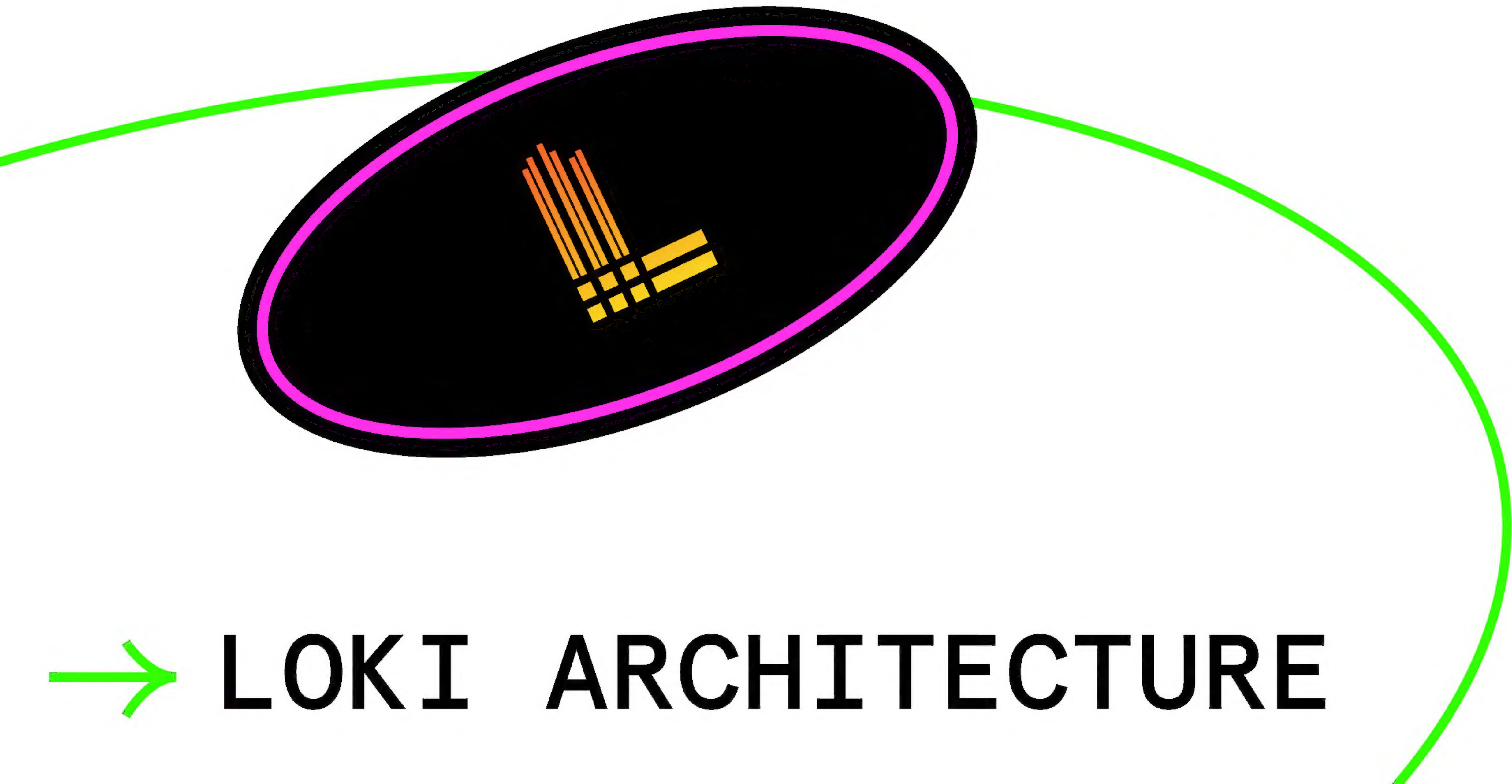
graylog



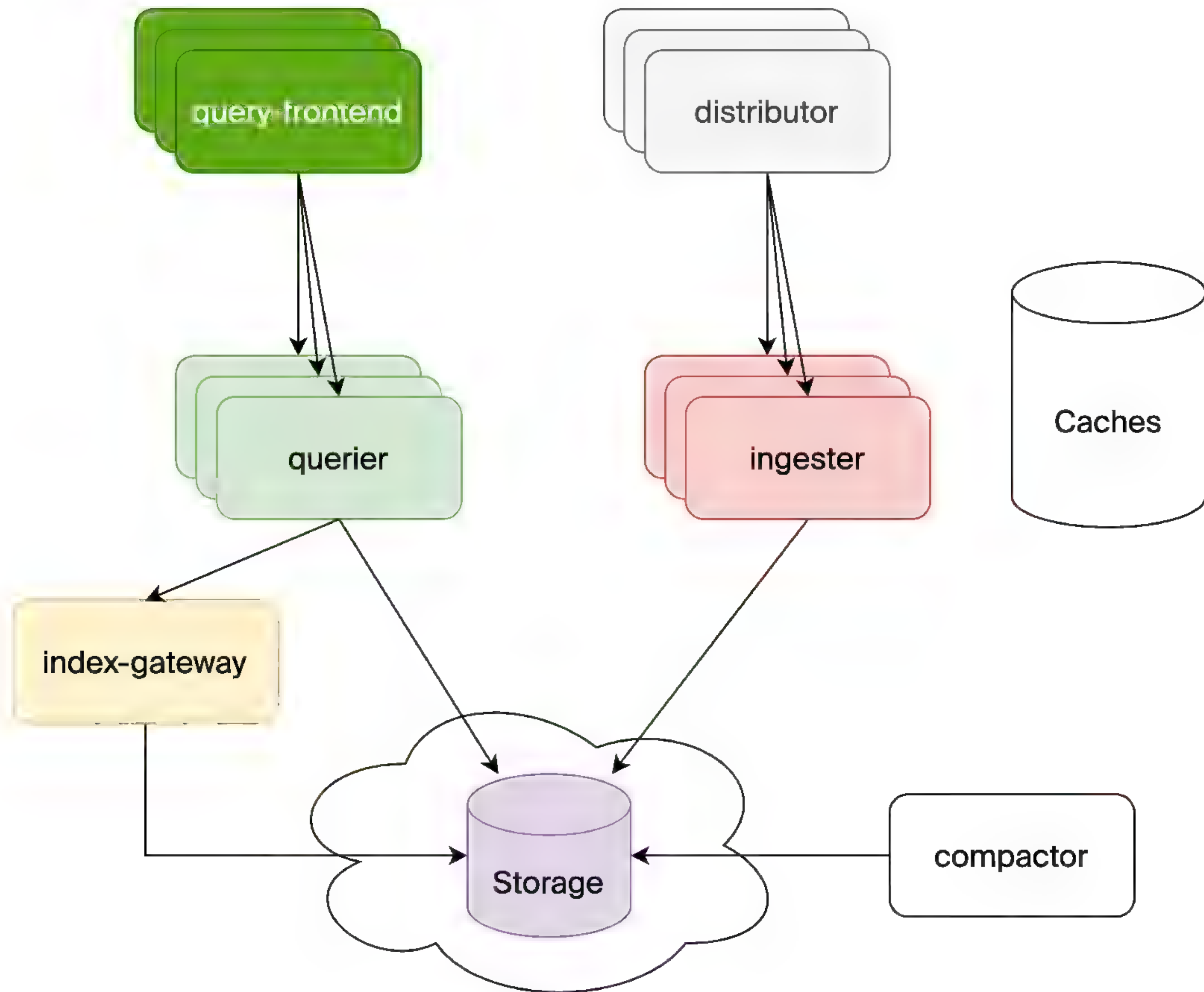
Grafana loki



ClickHouse



→ **LOKI ARCHITECTURE**



INDEXING

2019-12-11T10:01:02.123456789Z {app="nginx",cluster="us-west1"} GET /about

Timestamp
with nanosecond precision

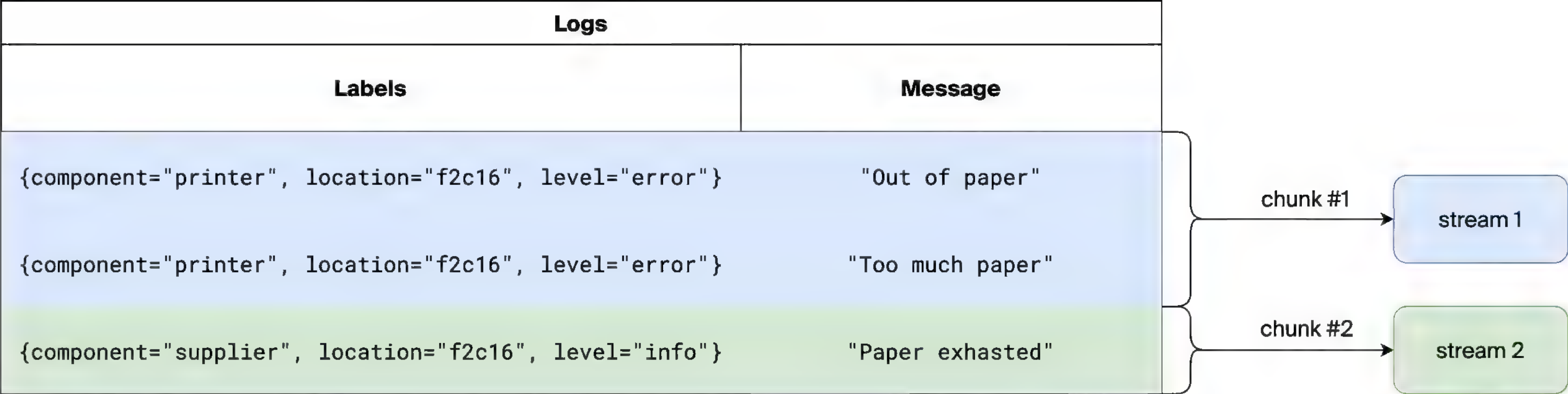
Prometheus-style Labels
key-value pairs

Content
logline

indexed

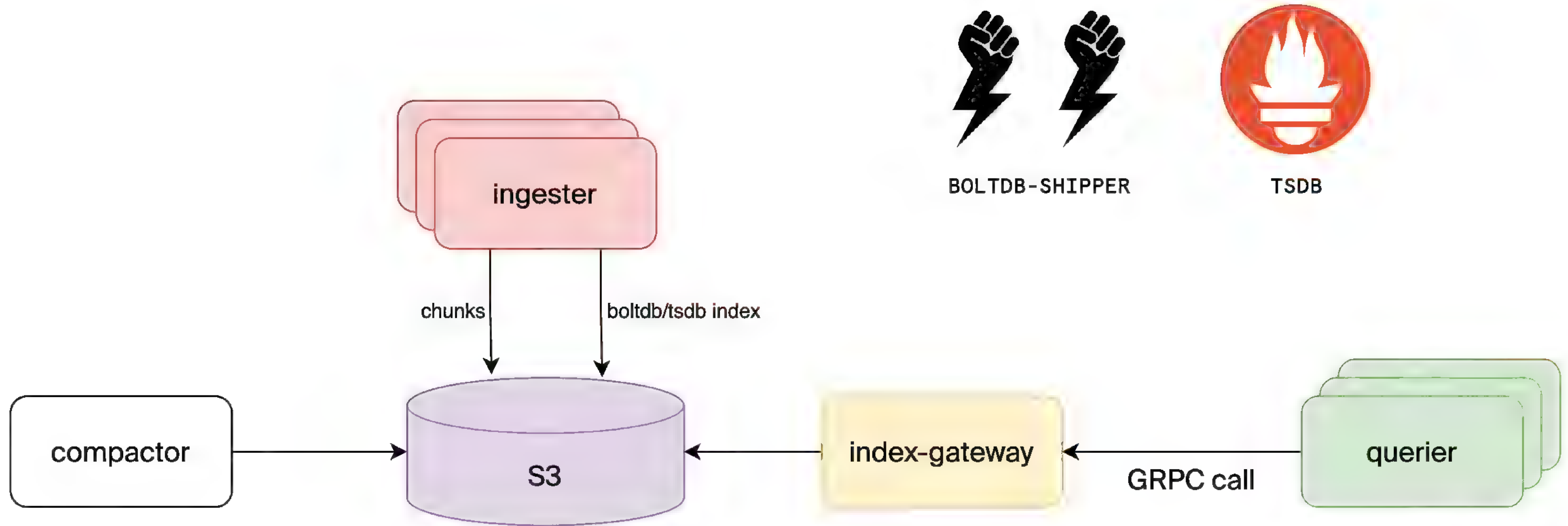
unindexed

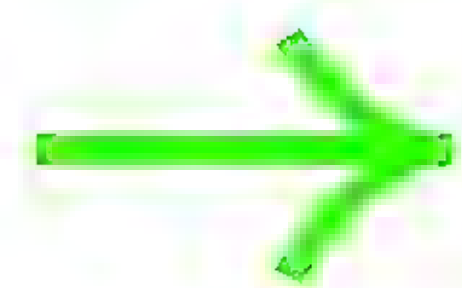
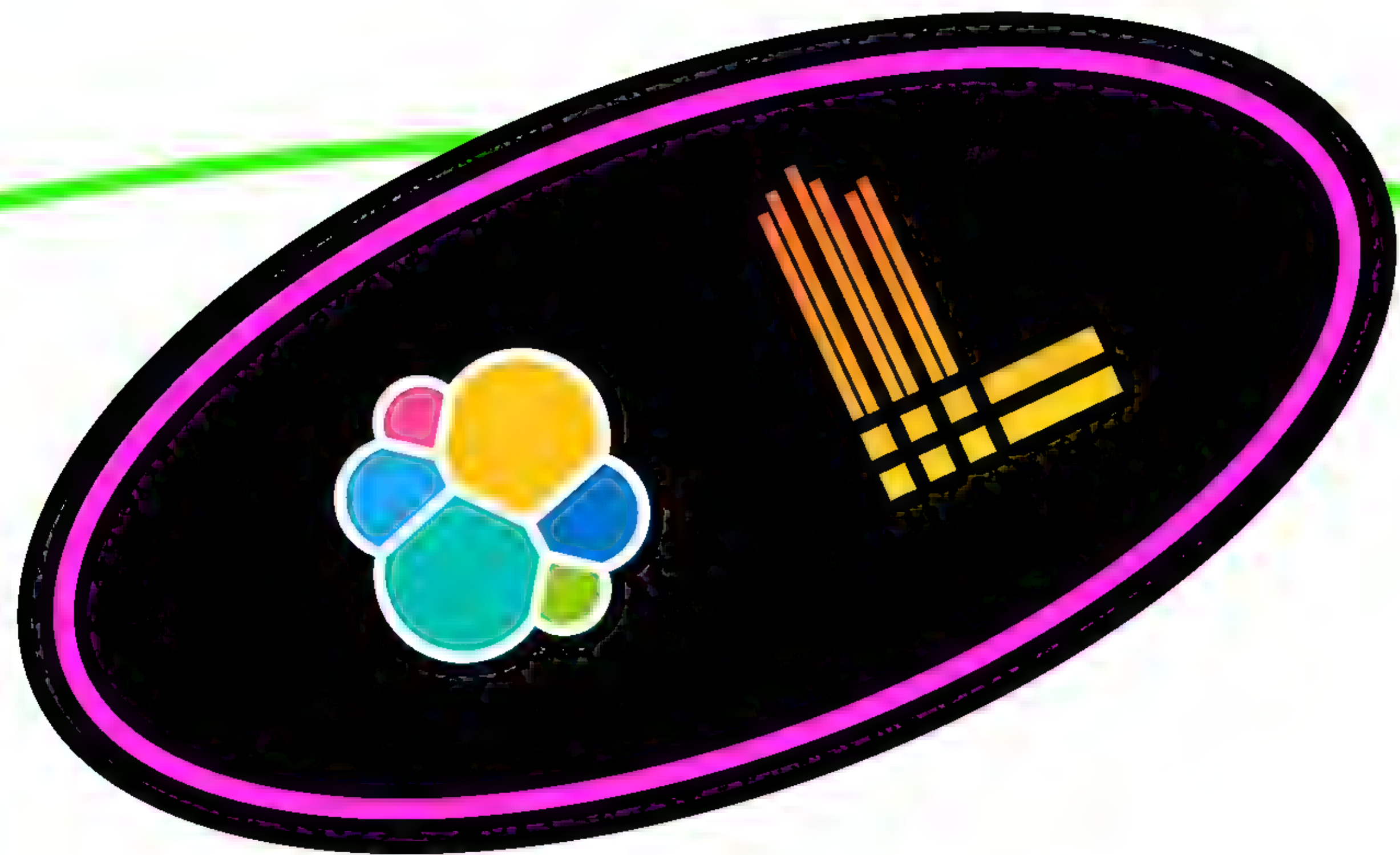
STREAMS



CHUNKS & INDEX

CHUNKS & INDEX → S3





ELASTICSEARCH → LOKI

**WHY WE DECIDED
TO MOVE?**

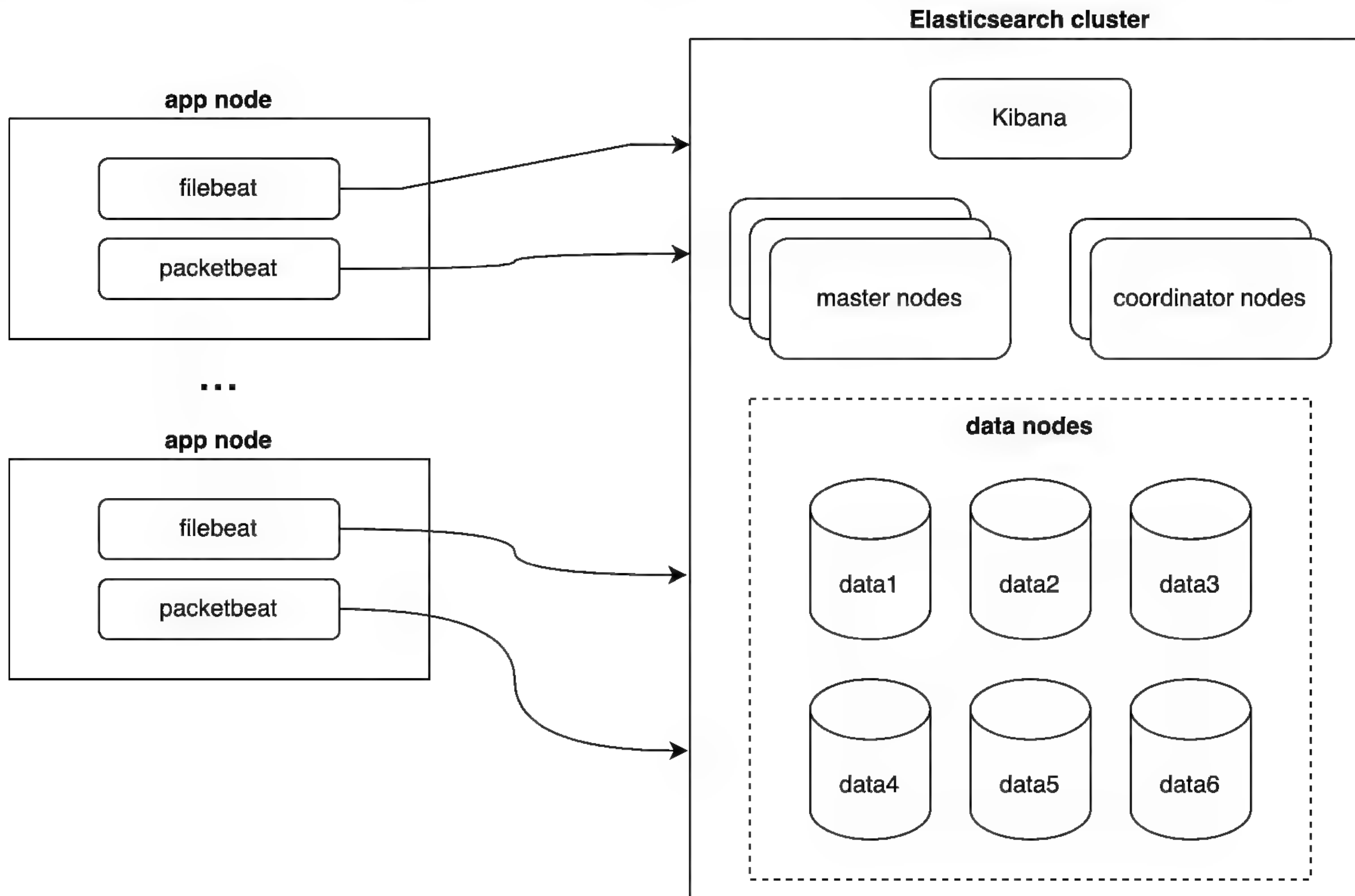


MOTIVATION TO DITCH ELASTIC

1 Huge money burden

2 Hard maintainability

ELASTICSEARCH-BASED INFRA



MIGRATION PLAN

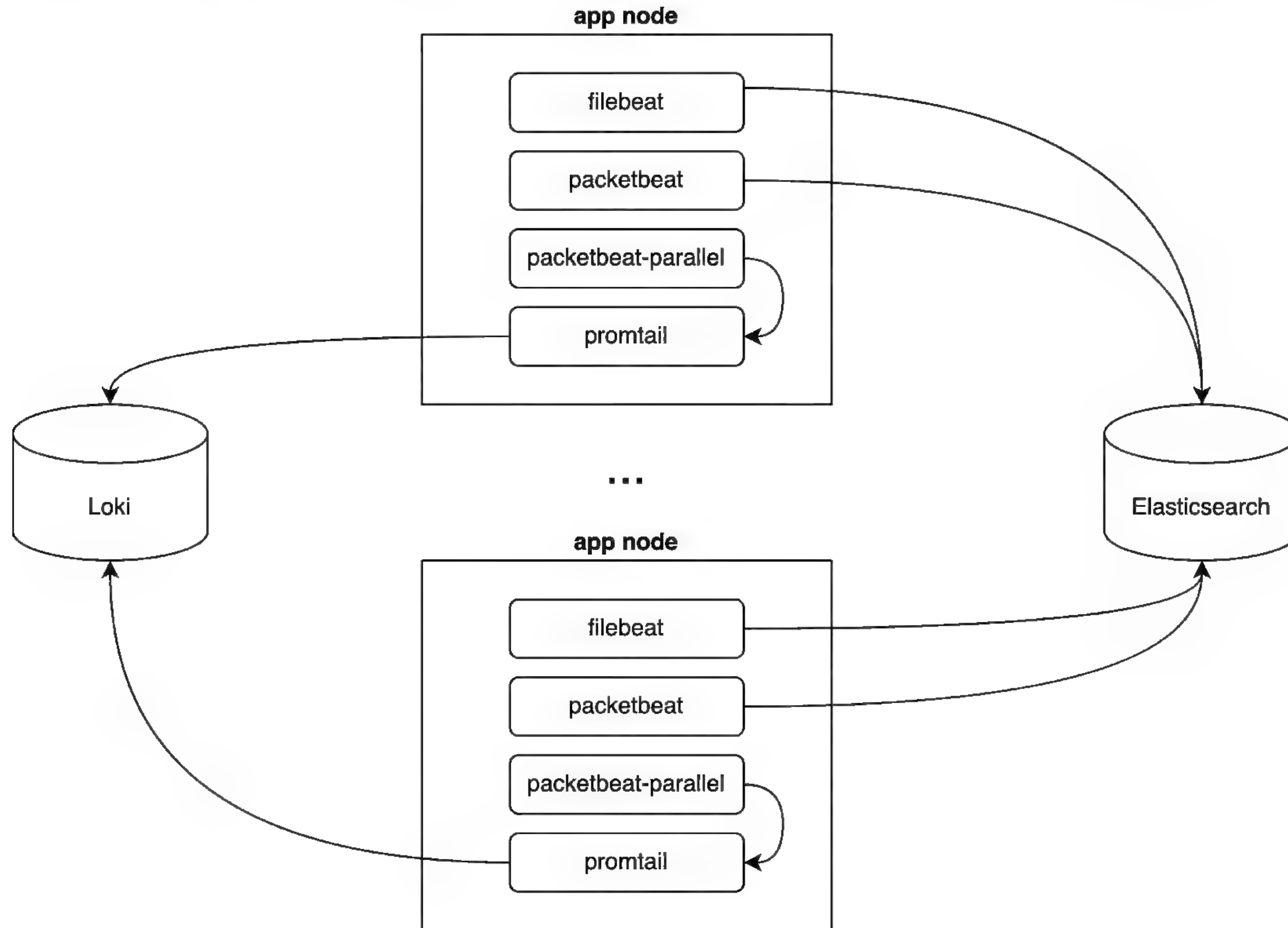
1. Deploy Loki cluster into Kubernetes
2. Deploy S3 storage
3. Start collecting live logs in Loki
4. Solve troubles, constantly reconfigure Loki to handle the load
5. Transfer old logs
6. Fail multiple times
7. Succeed, wait for the transfer to complete
8. Demolish Elasticsearch cluster

MIGRATION PLAN

1. Deploy Loki cluster into Kubernetes
2. Start collecting live logs in Loki
3. Transfer old logs

**COLLECTING
LIVE LOGS IN LOKI**

NOT BREAKING EVERYTHING



COLLECTING LOGS FROM FILES

- Application logs
- Packetbeat logs
- Postgres logs (new)
- 1C logs

```
scrape_configs:
- job_name: system
  static_configs:

- targets:
  - localhost
  labels:
    job: omni_services
    __path__: /var/log/omni/services/**/*.log

- targets:
  - localhost
  labels:
    job: packetbeat
    __path__: {{ promtail_packetbeat_folder }}/output

- targets:
  - localhost
  labels:
    job: postgres_logging
    __path__: {{ promtail_postgres_folder }}/**/*.log
```

Error

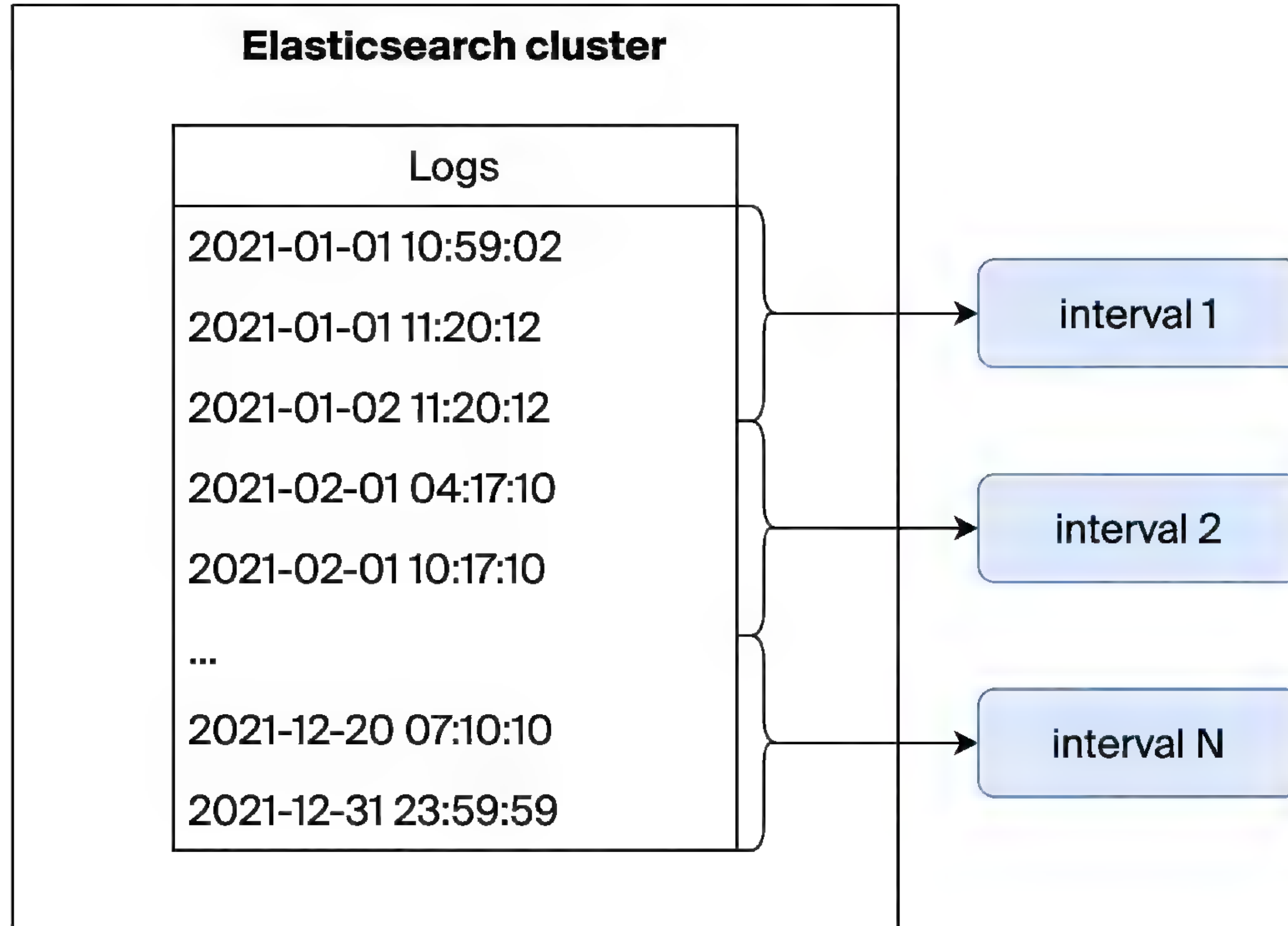
OK

TRANSFER OLD LOGS

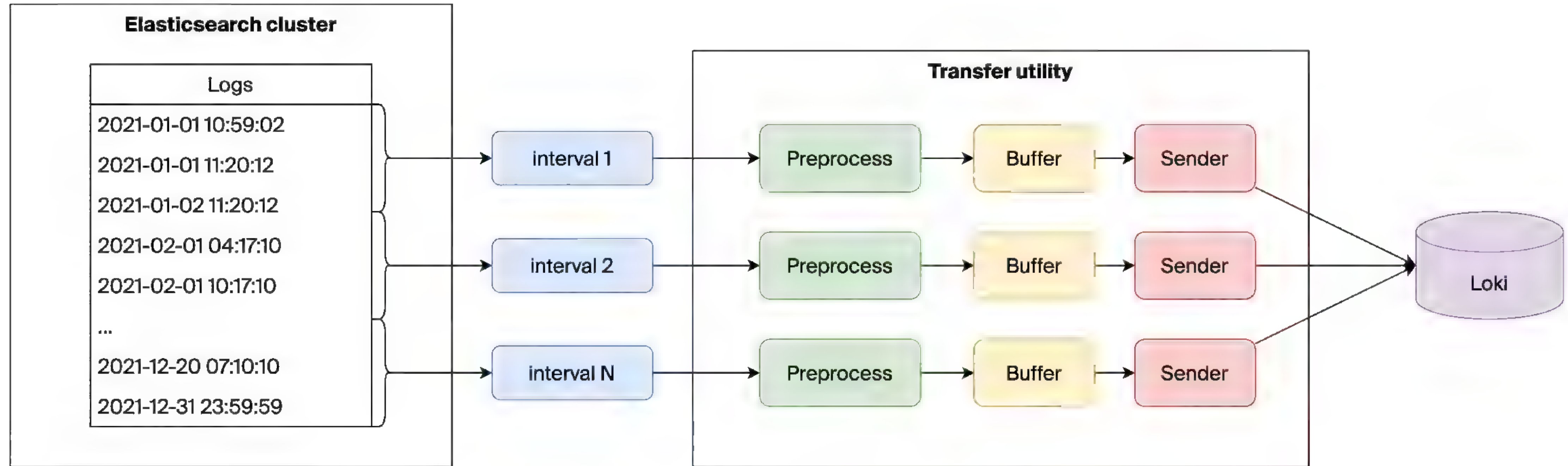
WHAT CONCERNED US?

- 1 year of log data
- 25 TB of storage

INITIAL ASSUMPTION



INITIAL ASSUMPTION



LOKI CONFIG

```
1 limits_config:
2   reject_old_samples: false
3
4   ingestion_rate_mb: 100
5   ingestion_burst_size_mb: 30
6   per_stream_rate_limit: "150MB"
7
8   max_streams_per_user: 0
9   max_global_streams_per_user: 0
10
11 retention_period: 8928h
```

TESTING OUR ASSUMPTION

entry too far behind, oldest
acceptable timestamp is:
2021-01-01T01:00:00

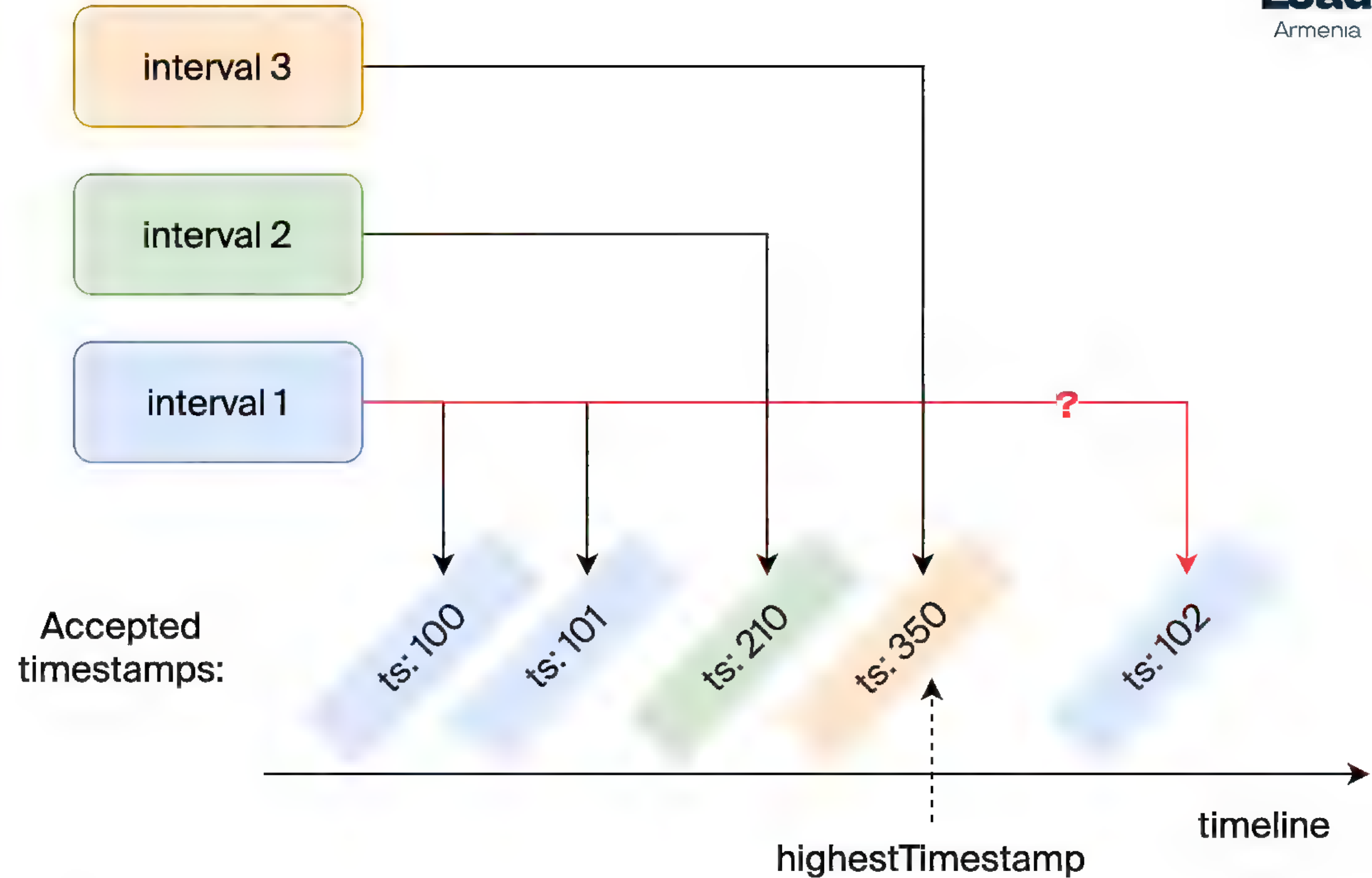
WHY IS THAT?

```
369 // The validity window for unordered writes is the highest timestamp present minus 1/2 * max-chunk-age.  
... 370 | cutoff := highestTs.Add(-s.cfg.MaxChunkAge / 2)  
371 if !isReplay && s.unorderedWrites && !highestTs.IsZero() && cutoff.After(entries[i].Timestamp) {  
372     failedEntriesWithError = append(failedEntriesWithError, entryWithError{&entries[i], chunkenc.ErrTooFarBehind(cutoff)})  
373     outOfOrderSamples++  
374     outOfOrderBytes += lineBytes  
375     continue  
376 }
```

This is the error



MIND EXPERIMENT



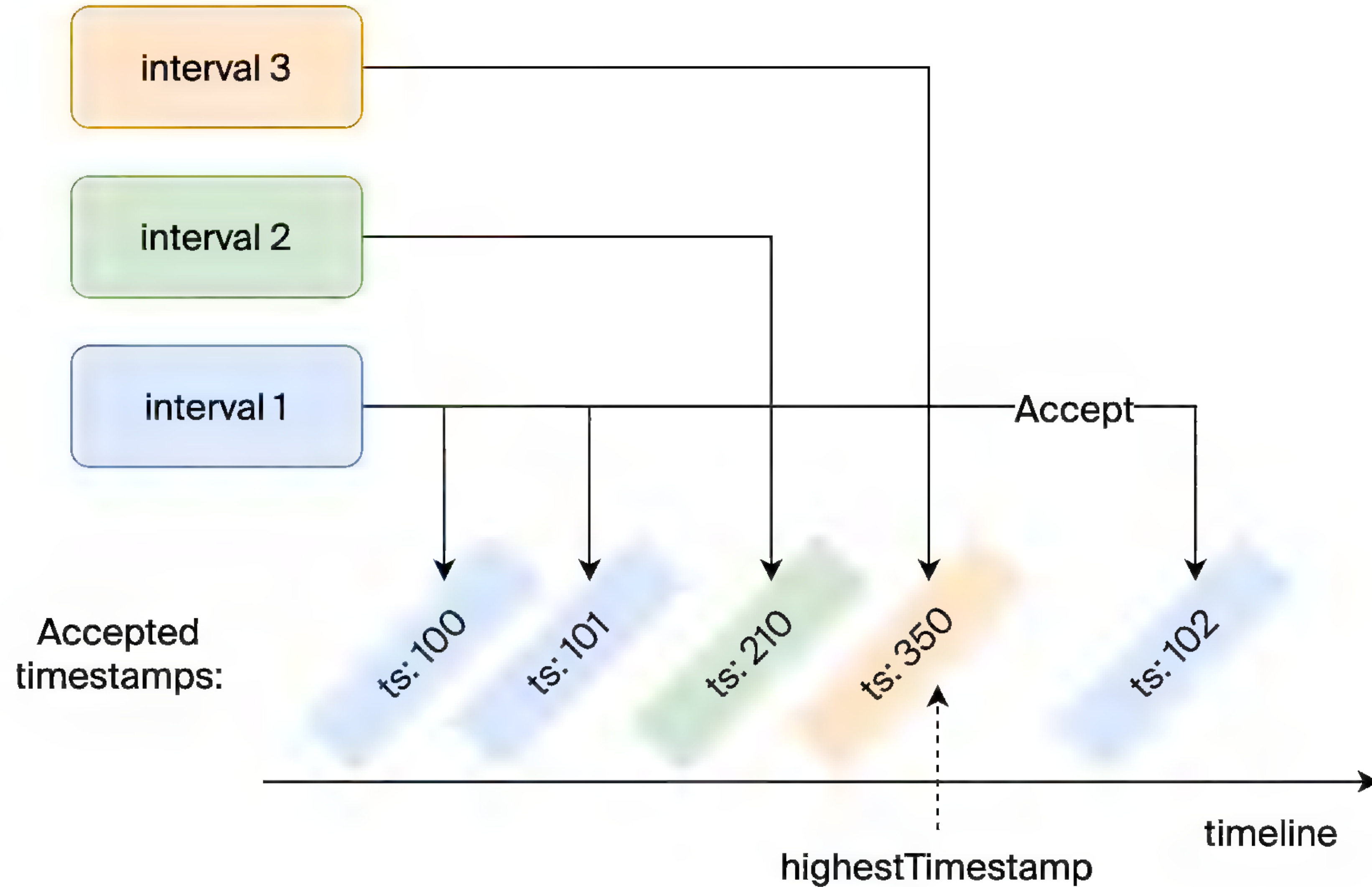
Minimal accepted timestamp is:

$$\text{highestTimestamp} - \text{maxChunkAge} / 2$$

TWEAK CONFIGURATION

```
1 ingestor:  
2     max_chunk_age: 8760h # 365d
```

SEEMS TO BE FIXED

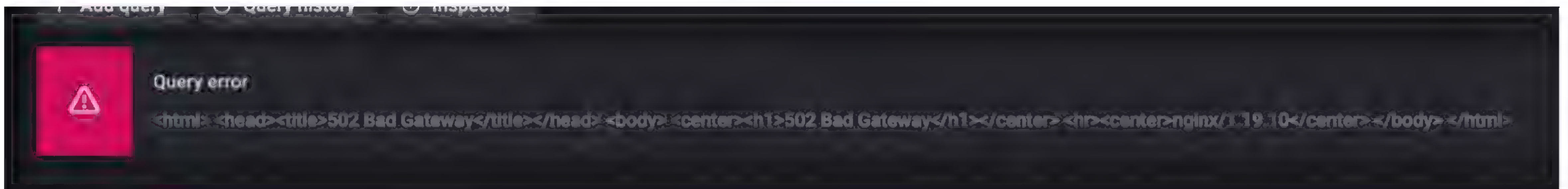
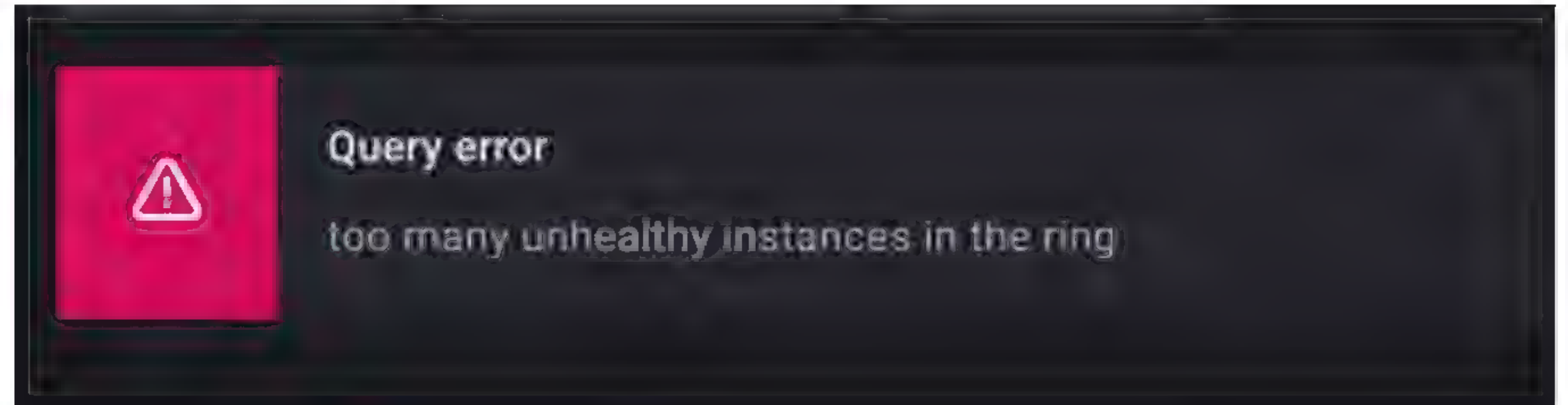
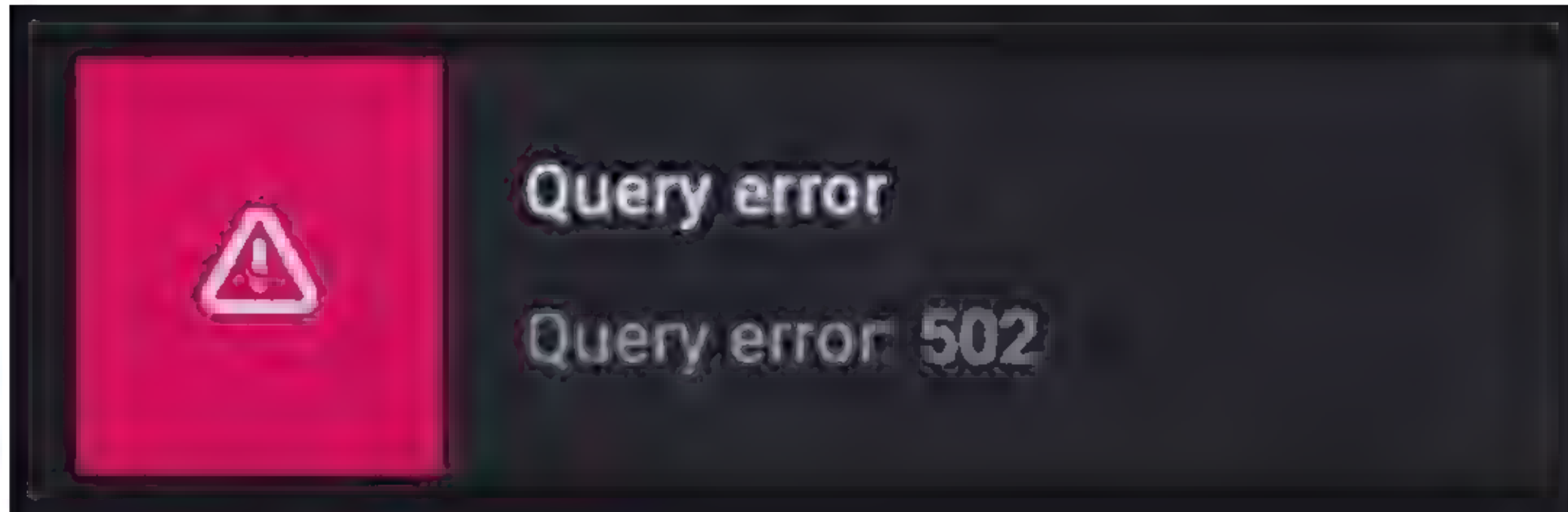


DID WE FOOL LOKI?








NOT REALLY

LOKI WON



WHAT IS HAPPENING?

- ✓ #5963 Context canceled error ✓ Closed 
- ⊙ #4015 Query from Grafana to Loki returning 502 when using 7 days ... ⊙ Open 
- ✓ #3524 Grafana "bad gateway" / Logcli "EOF", while frontend says "s..." ✓ Closed 
- ✓ #2540 Querier high memory demands ✓ Closed 
- ✓ #3753 [loki distributed] slow query (high cpu usage) and time out ✓ Closed 

It is time to stop guessing and start thinking

BLOCK & CHUNK

ts (varint)	len (uvarint)	log-1 bytes
ts (varint)	len (uvarint)	log-2 bytes
ts (varint)	len (uvarint)	log-3 bytes
ts (varint)	len (uvarint)	log-n bytes

MagicNumber(4b)	version(1b)
block-1 bytes	checksum (4b)
block-2 bytes	checksum (4b)
block-n bytes	checksum (4b)
#blocks (uvarint)	
#entries(uvarint) mint, maxt (varint) offset, len (uvarint)	
#entries(uvarint) mint, maxt (varint) offset, len (uvarint)	
#entries(uvarint) mint, maxt (varint) offset, len (uvarint)	
#entries(uvarint) mint, maxt (varint) offset, len (uvarint)	
checksum(from #blocks)	
metasOffset - offset to the point with #blocks	

BLOCK & CHUNK

ts (varint)	len (uvarint)	log-1 bytes
ts (varint)	len (uvarint)	log-2 bytes
ts (varint)	len (uvarint)	log-3 bytes
ts (varint)	len (uvarint)	log-n bytes

MagicNumber(4b)	version(1b)
block-1 bytes	checksum (4b)
block-2 bytes	checksum (4b)
block-n bytes	checksum (4b)
#blocks (uvarint)	
#entries(uvarint)	mint, maxt (varint) offset, len (uvarint)
#entries(uvarint)	mint, maxt (varint) offset, len (uvarint)
#entries(uvarint)	mint, maxt (varint) offset, len (uvarint)
#entries(uvarint)	mint, maxt (varint) offset, len (uvarint)
checksum(from #blocks)	
metasOffset - offset to the point with #blocks	

INVESTIGATION STAGE

```
302         for i := 0; i < len(entries); i++ {
303             chunk := &s.chunks[len(s.chunks)-1]
304             if chunk.closed || !chunk.chunk.SpaceFor(&entries[i]) || s.cutChunkForSynchronization(i) {
305                 chunk = s.cutChunk(ctx)
306             }
307
308             chunk.lastUpdated = time.Now()
... 309             if err := chunk.chunk.Append(&entries[i]); err != nil {
310                 invalid = append(invalid, entryWithError(&entries[i], err))
311                 if chunkenc.IsOutOfOrderErr(err) {
312                     outOfOrderSamples++
313                     outOfOrderBytes += len(entries[i].Line)
314                 }
315                 continue
316             }
```

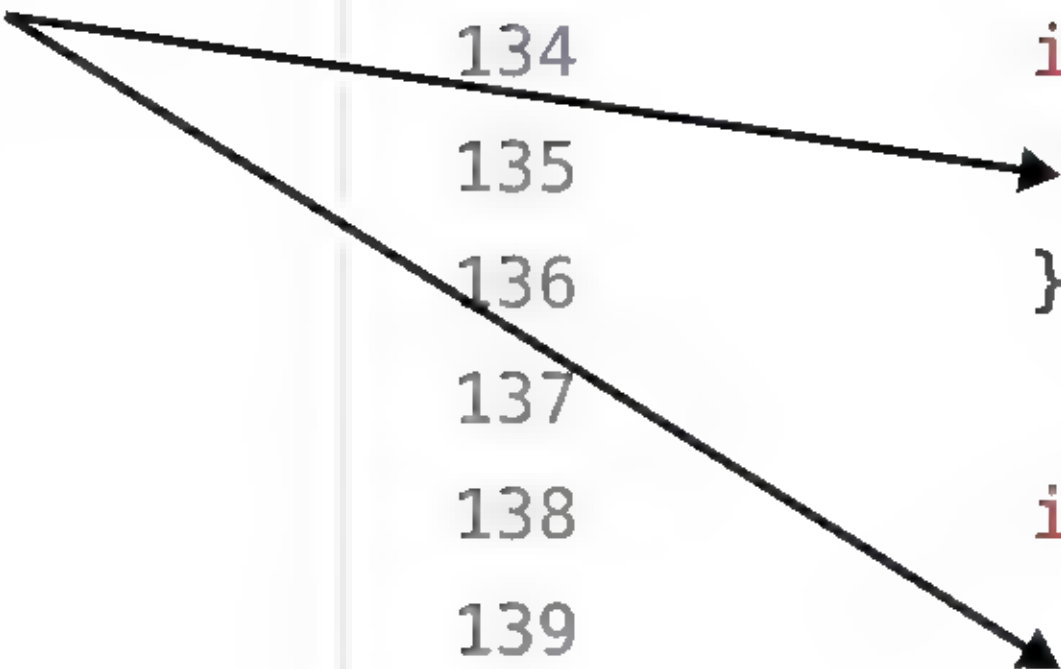

INVESTIGATION STAGE

```
...
667 // Append implements Chunk.
668 func (c *MemChunk) Append(entry *logproto.Entry) error {
669     entryTimestamp := entry.Timestamp.UnixNano()
670
671     // If the head block is empty but there are cut blocks, we have to make
672     // sure the new entry is not out of order compared to the previous block
673     if c.headFmt < UnorderedHeadBlockFmt && c.head.IsEmpty() && len(c.blocks) > 0 && c.blocks[len(c.blocks)-1].maxt > entryTimestamp {
674         return ErrOutOfOrder
675     }
676
... 677 |     if err := c.head.Append(entryTimestamp, entry.Line); err != nil {
678         return err
679     }
680
681     if c.head.UncompressedSize() >= c.blockSize {
682         return c.cut()
683     }
684
685     return nil
686 }
```

INVESTIGATION STAGE

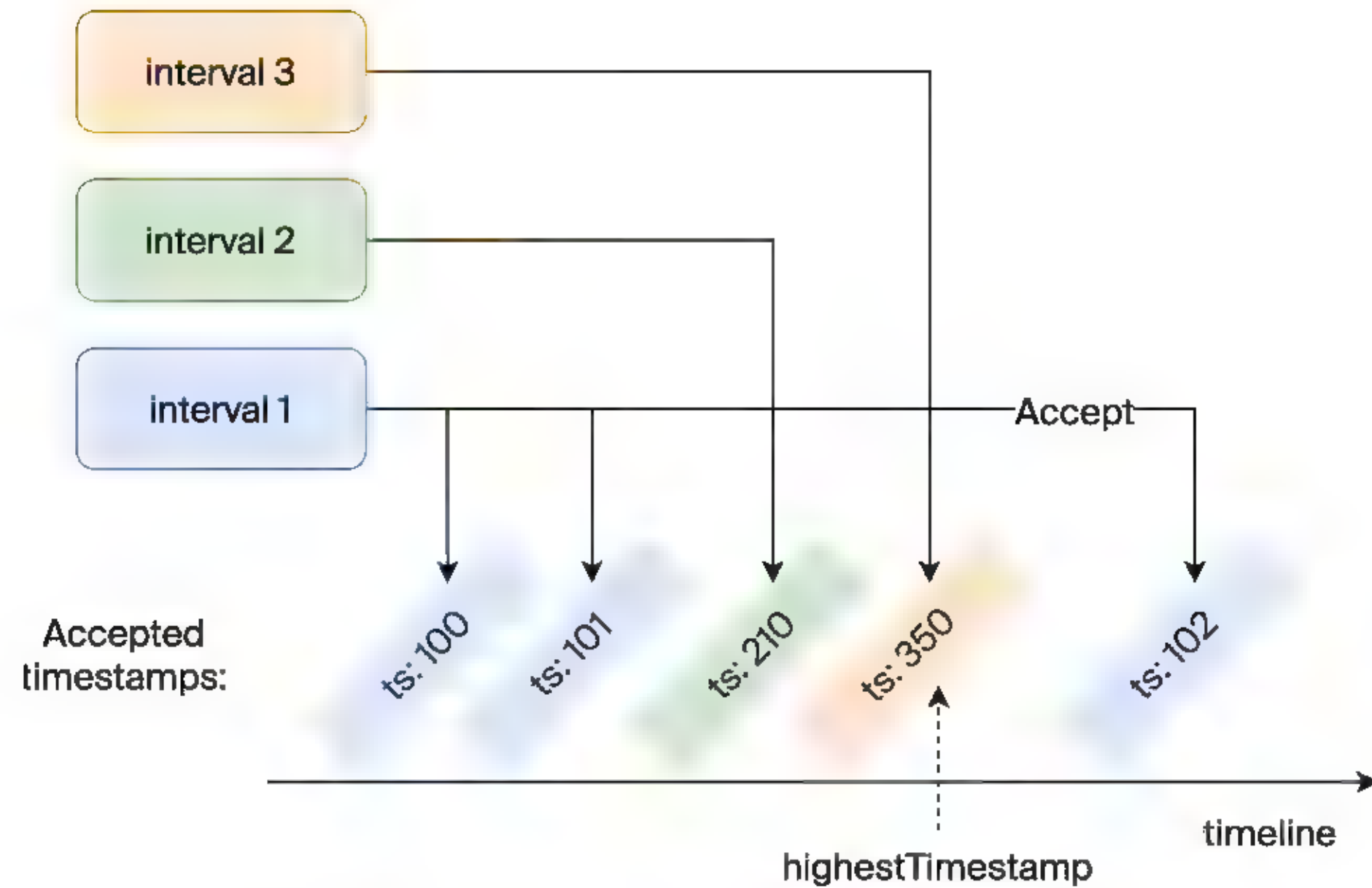
It just
updates the
boundaries

```
104 func (hb *unorderedHeadBlock) Append(ts int64, line string) error {  
    . . .  
133 // Update hb metadata  
134 if hb.size == 0 || hb.mint > ts {  
135     hb.mint = ts  
136 }  
137  
138 if hb.maxt < ts {  
139     hb.maxt = ts  
140 }  
141  
142 hb.size += len(line)  
143 hb.lines++  
144  
145 return nil  
146 }
```



SO WHAT?

BLOCKS MIX UP



Head Block

mint	100
maxt	350

ts: 100

ts: 101

ts: 210

ts: 350

ts: 103

Chunk

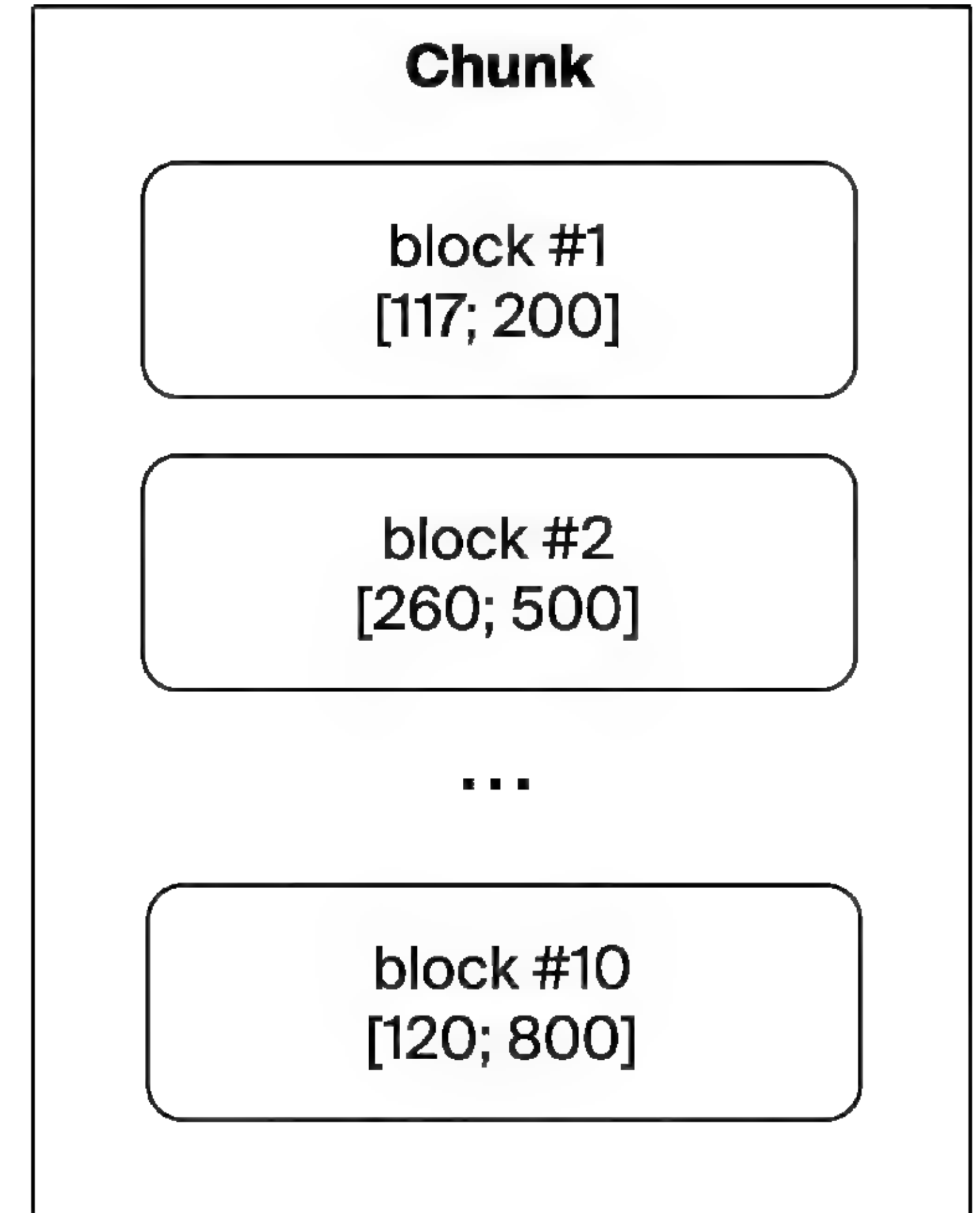
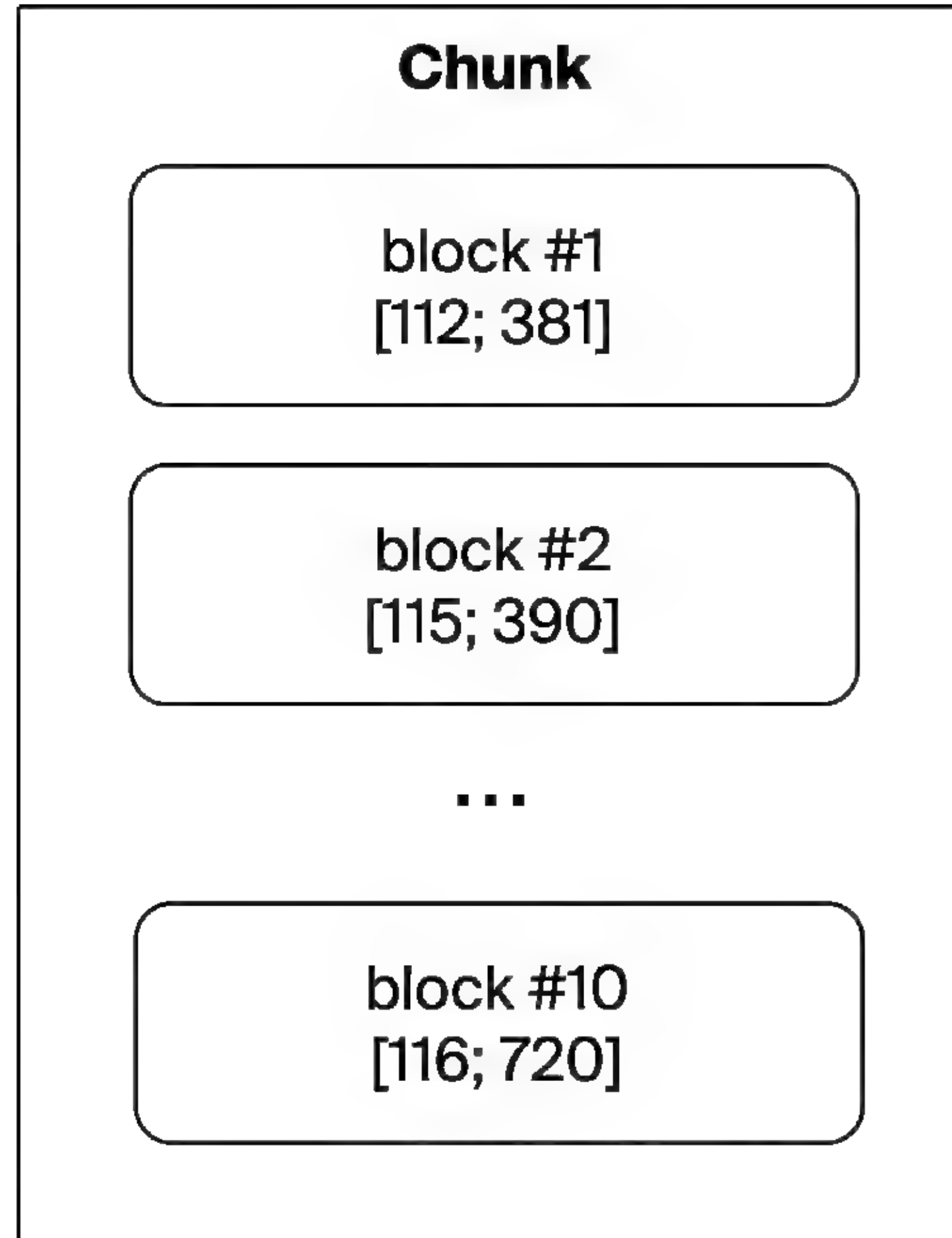
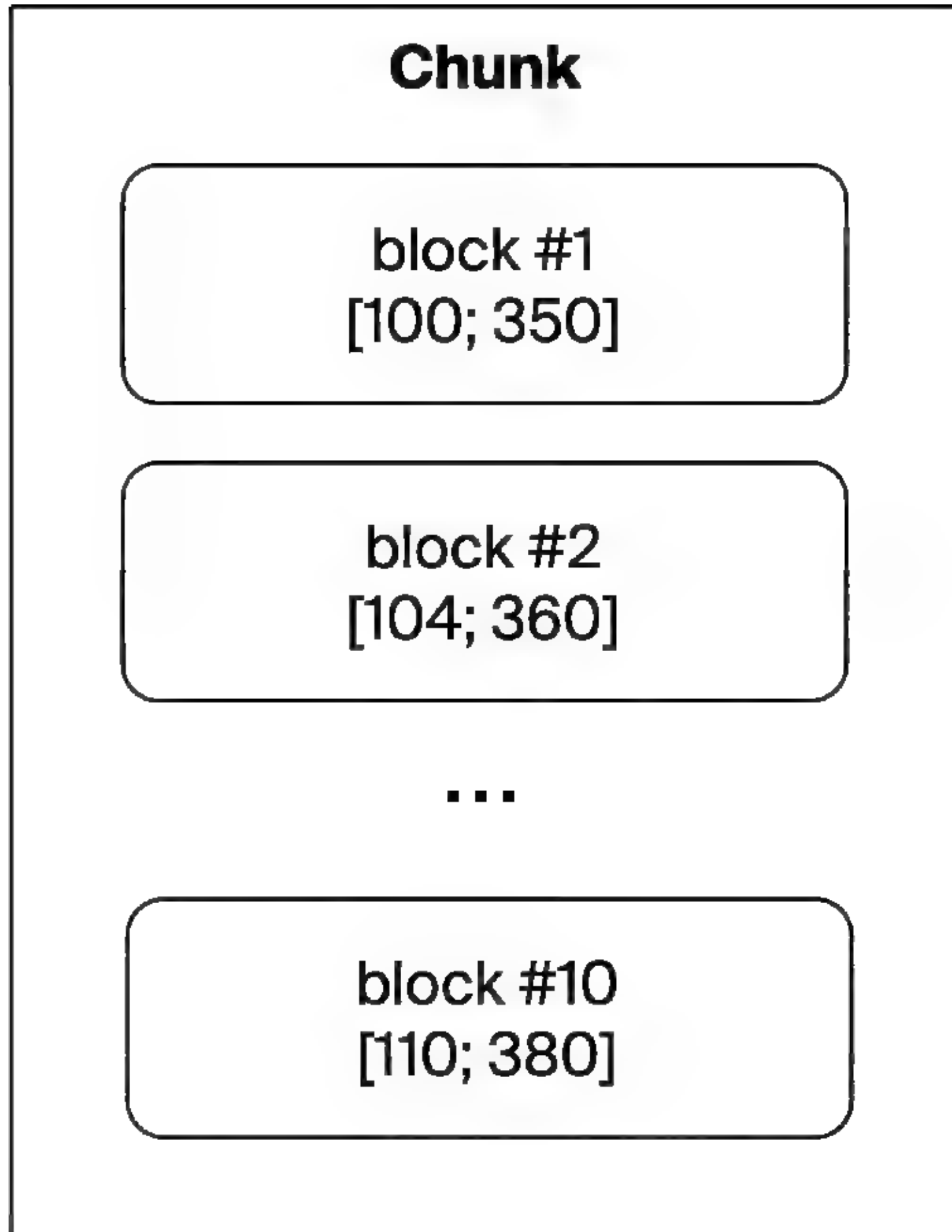
block #1
[100; 350]

block #2
[104; 360]

...

block #10
[110; 380]

CHUNKS MIX UP AS WELL



QUERY LOGS IN [120; 200]

Chunk

block #1
[100; 350]

block #2
[104; 360]

...

block #10
[110; 380]

Chunk

block #1
[112; 381]

block #2
[115; 390]

...

block #10
[116; 720]

Chunk

block #1
[117; 200]

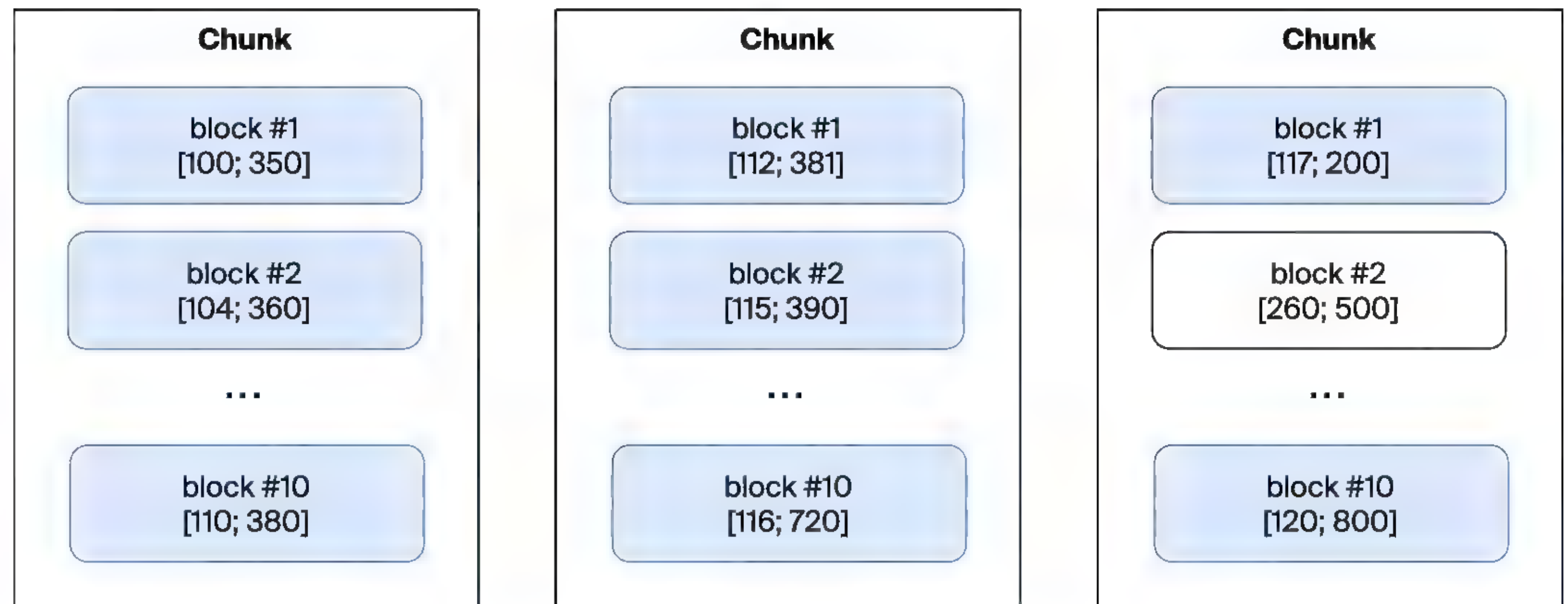
block #2
[260; 500]

...

block #10
[120; 800]

QUERY LOGS IN [120; 200]

- Loki is forced to download almost all chunks from the storage
- Process in memory
- Resort & filter

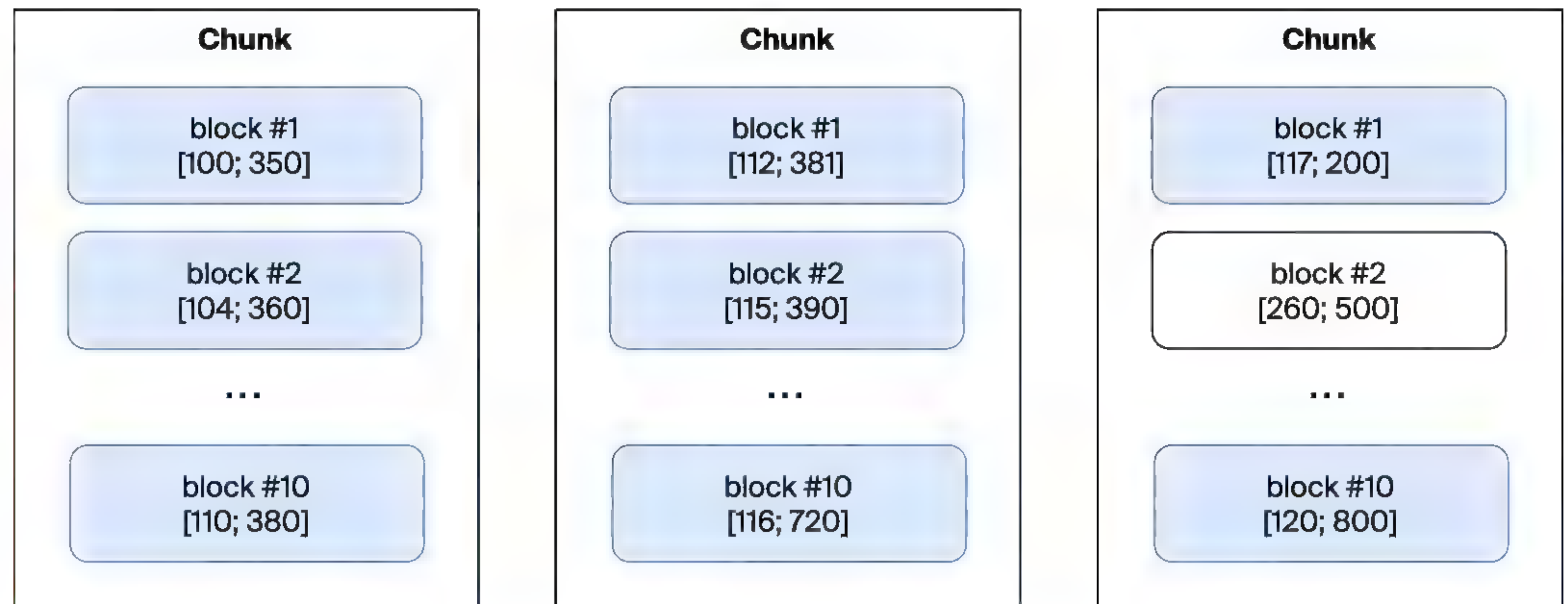


QUERY LOGS IN [120; 200]

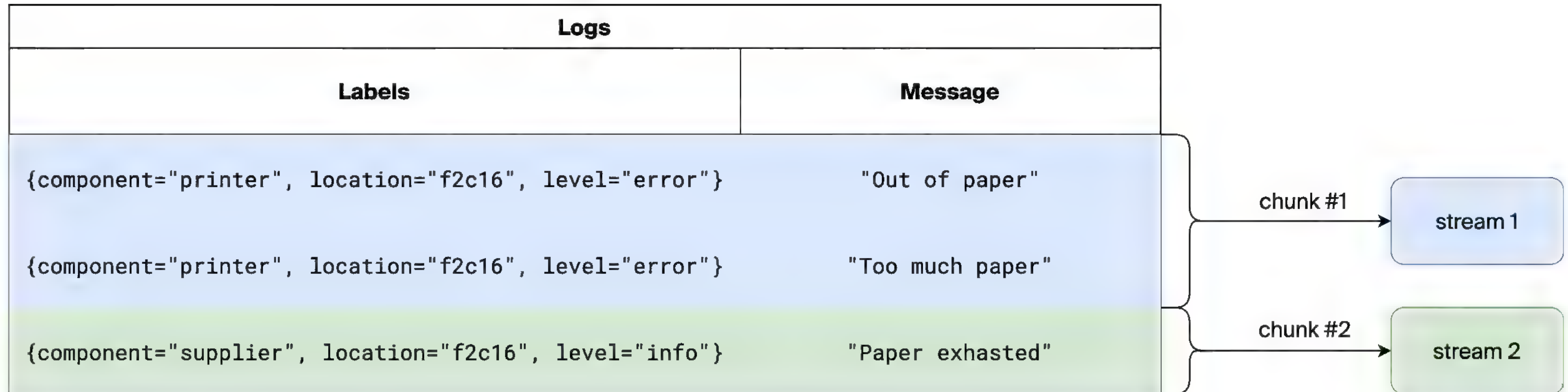
- Loki is forced to download almost all chunks from the storage
- Process in memory
- Resort & filter



<https://github.com/grafana/loki/issues/2540>



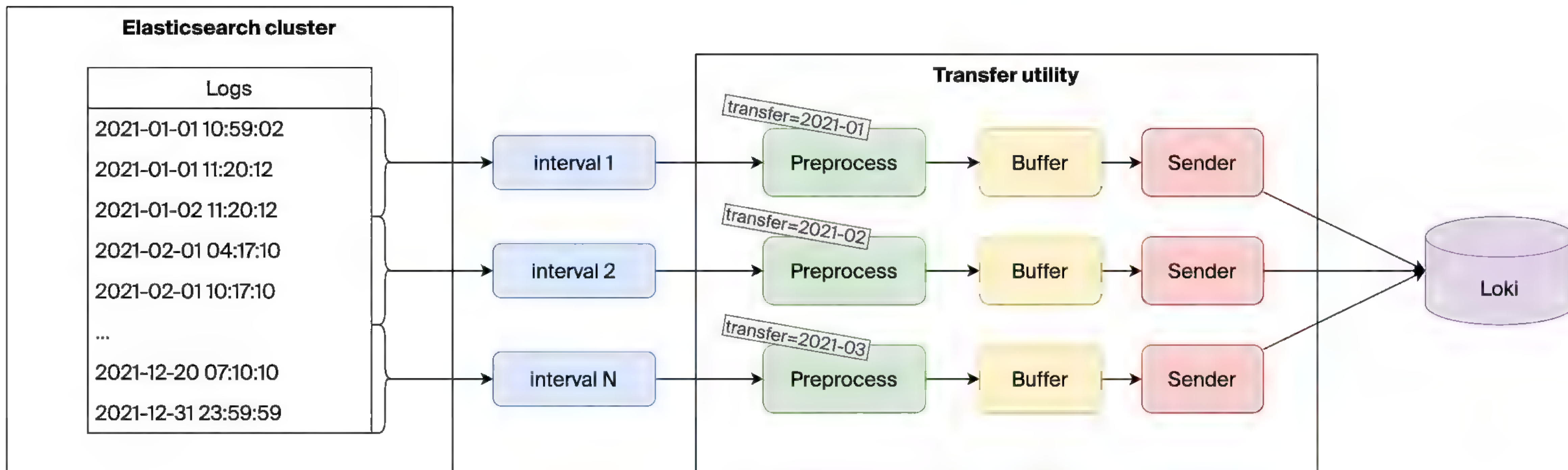
HOW TO FIX EVERYTHING?



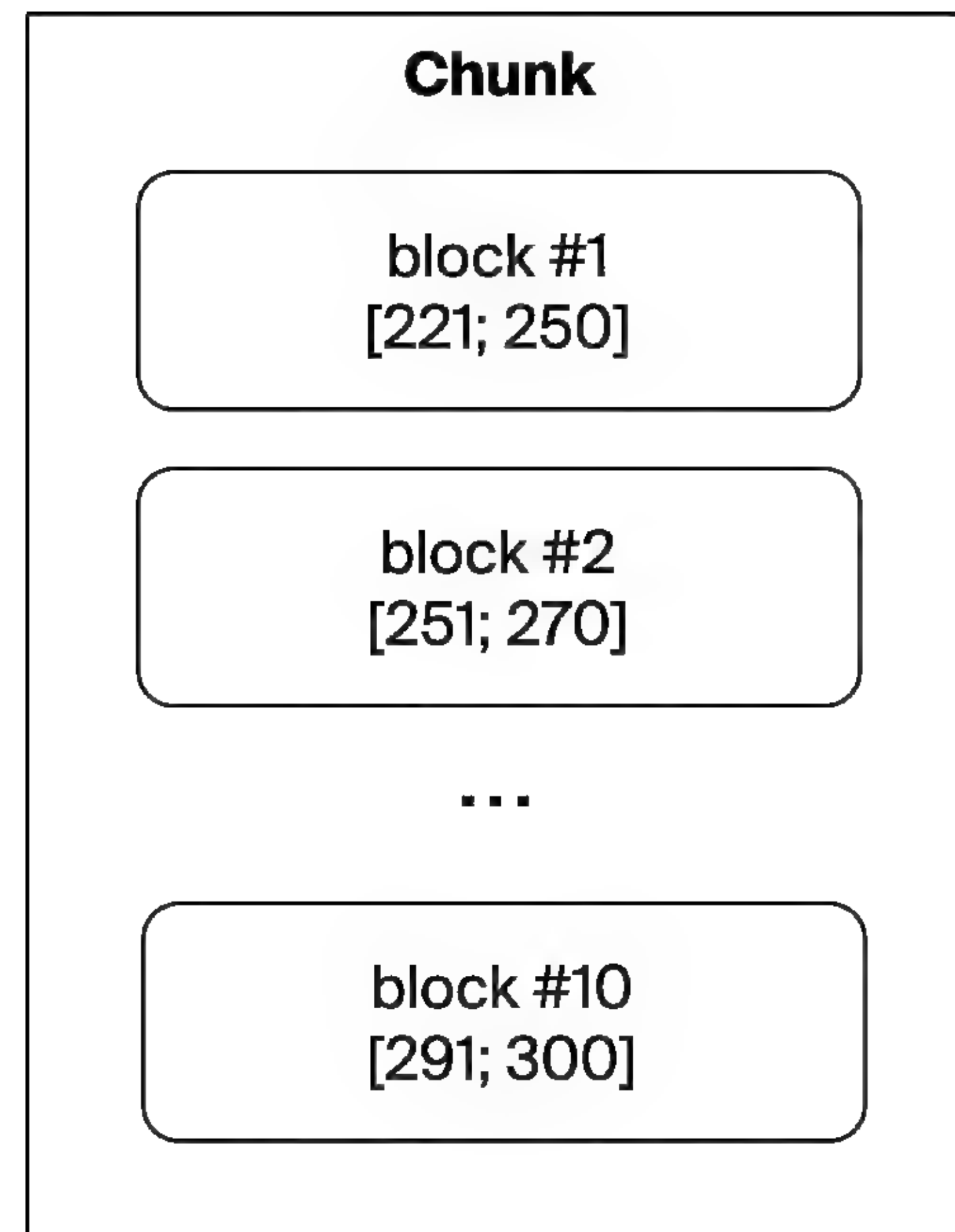
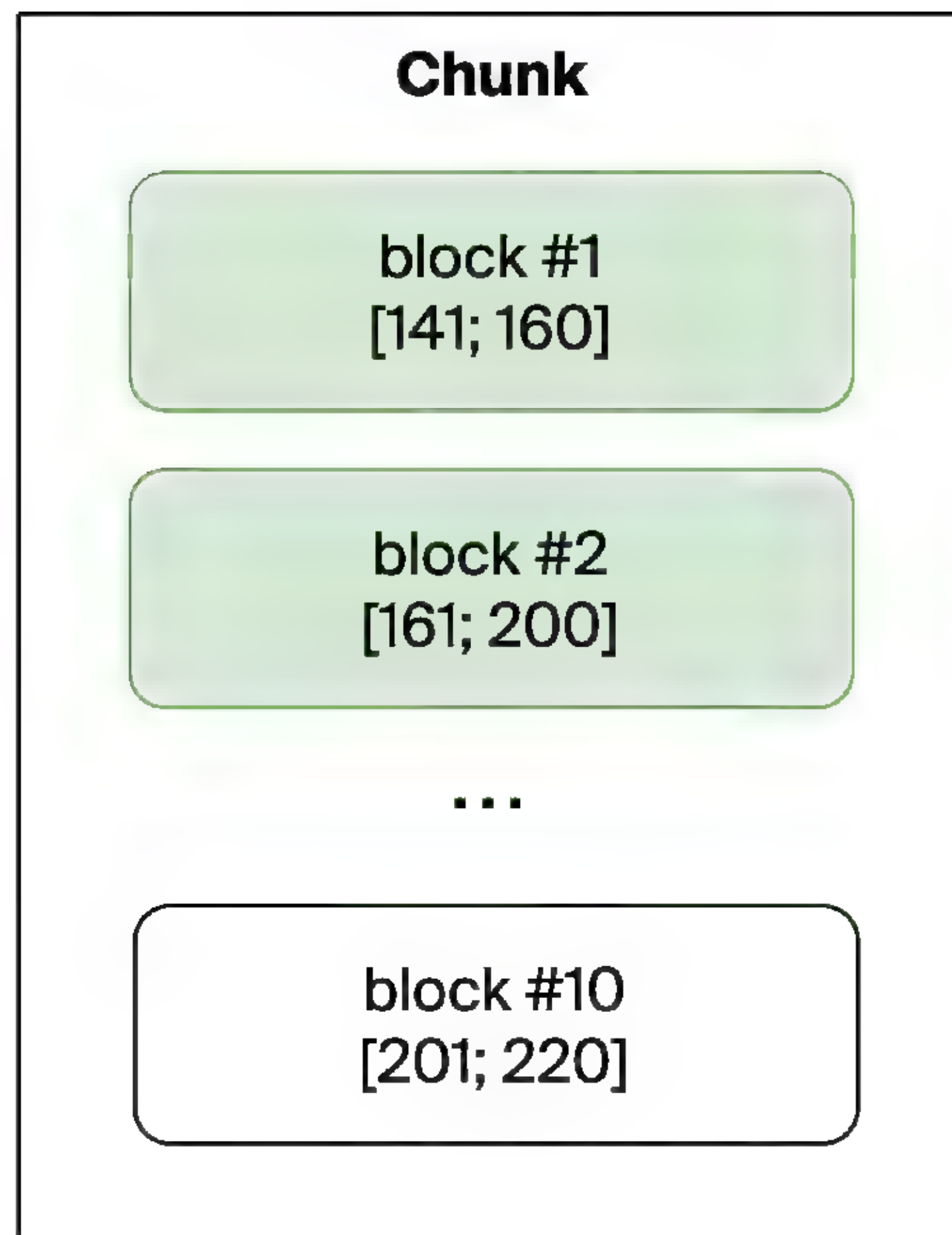
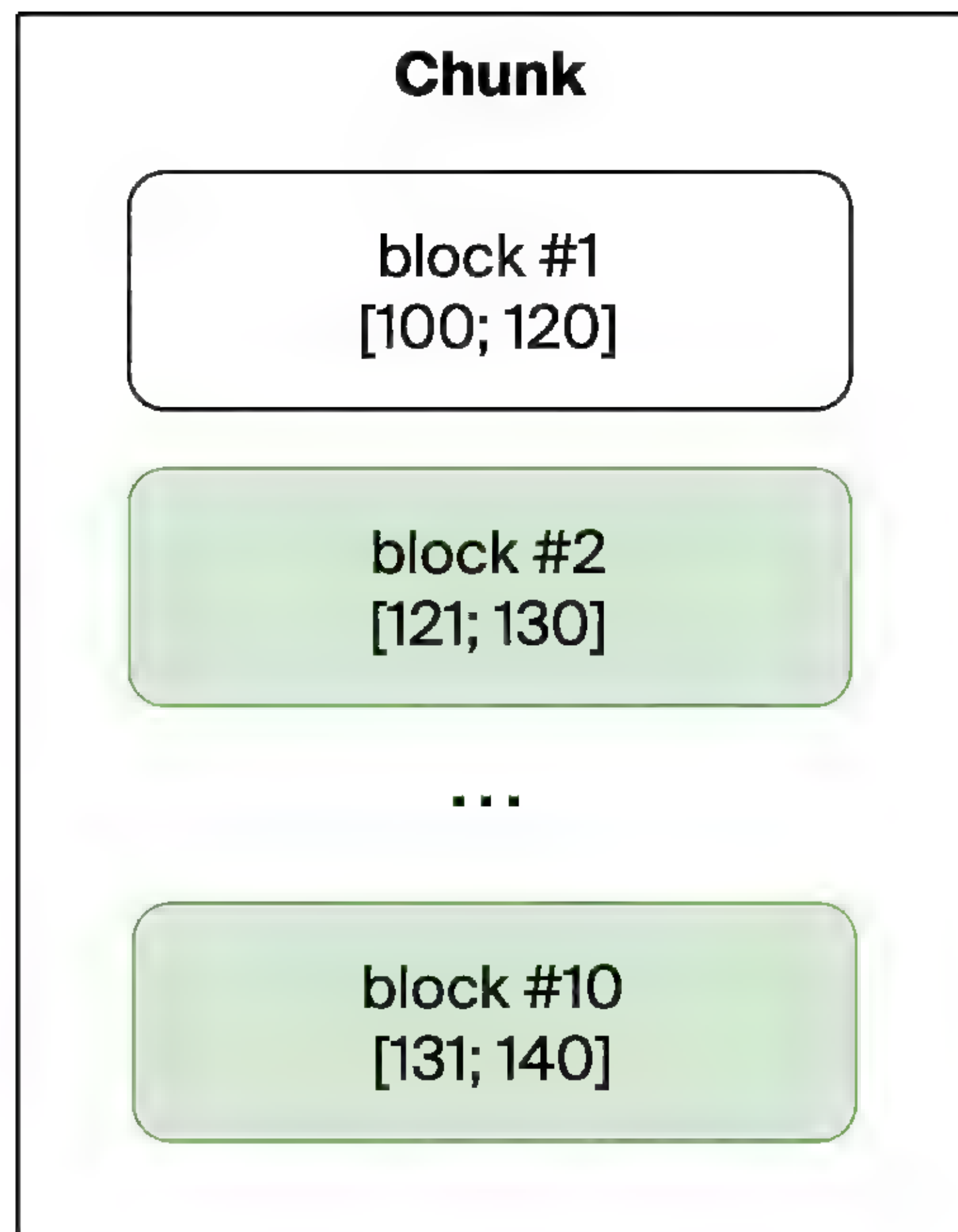
HOW TO FIX EVERYTHING?

1. Don't set **max_chunk_age** to a high value
2. Don't send logs in **parallel** for the same set of labels (i.e. a stream)
3. You may create **additional streams** by adding new labels
4. Send logs within a stream only in the **strict order**

JUST ADD LABELS

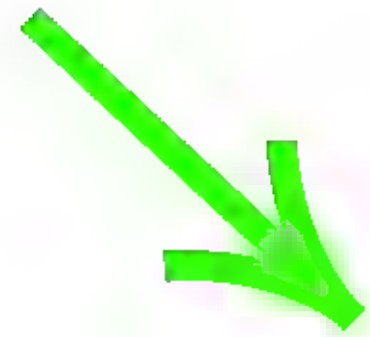


QUERY LOGS IN [120; 200]

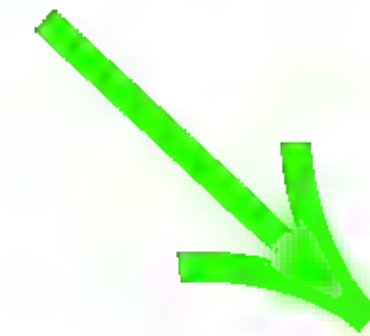


CHUNKS SITUATION

1 Query matches less chunks



2 All blocks within all chunks are strictly ordered

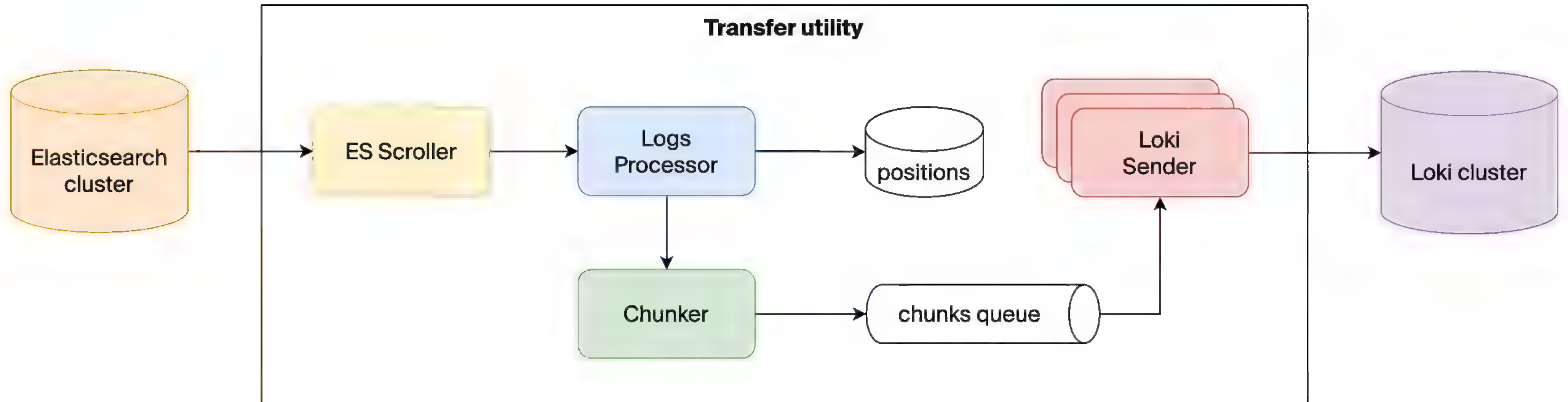


3 Fast and efficient query execution

TRANSFER IMPLEMENTATION



OK
GO



ES SCROLLER

- 1 Sort by (@timestamp, log.offset)
- 2 Use search_after to scroll & resume search
- 3 Do not let a consumer wait for data
- 4 Handle errors properly
- 5 Some shards may return no data

ALWAYS WAIT FOR ERROR RESOLUTION – WE CANNOT LOOSE LOG LINES

LOGS PROCESSOR

PERFORMS A SIMILAR PROCESSING AS PROMTAIL

- 1 Extract timestamp
- 2 Extract labels
- 3 Filter logs
- 4 Make a unique stream to ensure parallelism

LOGS PROCESSOR

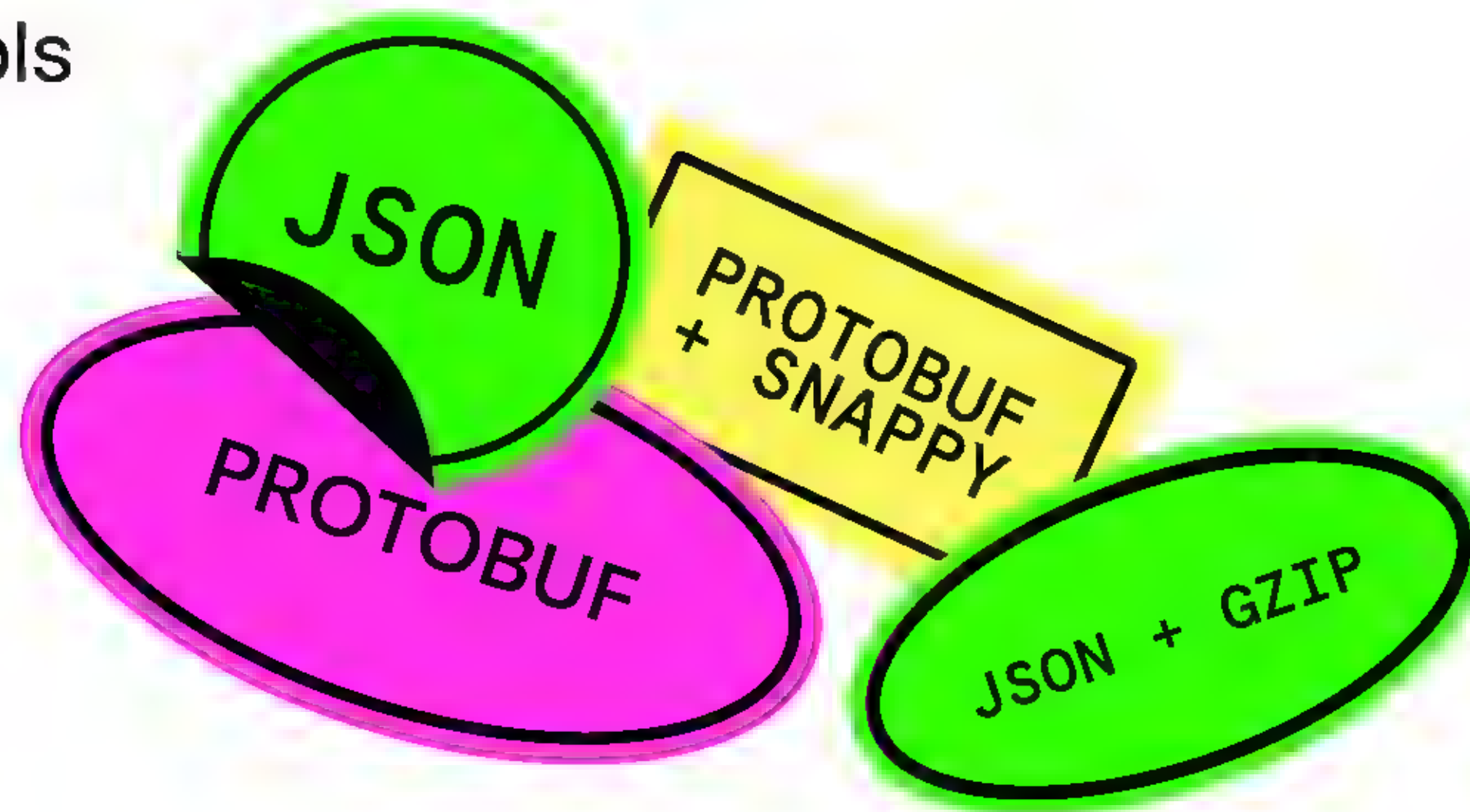
```
class Transfer(BaseTransfer):  
    def extract_doc_labels(self, source: dict) → Optional[MutableMapping[str, str]]:  
        return dict(  
            app=source.get("fields", {}).get("service_name"),  
            job="logs",  
            level=source.get("level"),  
            node_name=source.get("host", {}).get("name"),  
            logger_name=source.get("logger_name"),  
        )
```


LOKI SENDER

- 1 Send data to /loki/api/v1/push
- 2 Send data sequentially
- 3 Use only 1 worker (for now)
- 4 Support for different protocols

LOKI SENDER

- 1 Send data to /loki/api/v1/push
- 2 Send data sequentially
- 3 Use only 1 worker (for now)
- 4 Support for different protocols



YOU CAN ALREADY USE IT



[HTTPS://GITHUB.COM/KTSSTUDIO/ES2LOKI](https://github.com/KTSSTUDIO/ES2LOKI)

YOU CAN ALREADY USE IT

```
class Transfer(BaseTransfer):  
    def extract_doc_labels(self, source: dict) → Optional[MutableMapping[str, str]]:  
        return dict(  
            app=source.get("fields", {}).get("service_name"),  
            job="logs",  
            level=source.get("level"),  
            node_name=source.get("host", {}).get("name"),  
            logger_name=source.get("logger_name"),  
        )
```

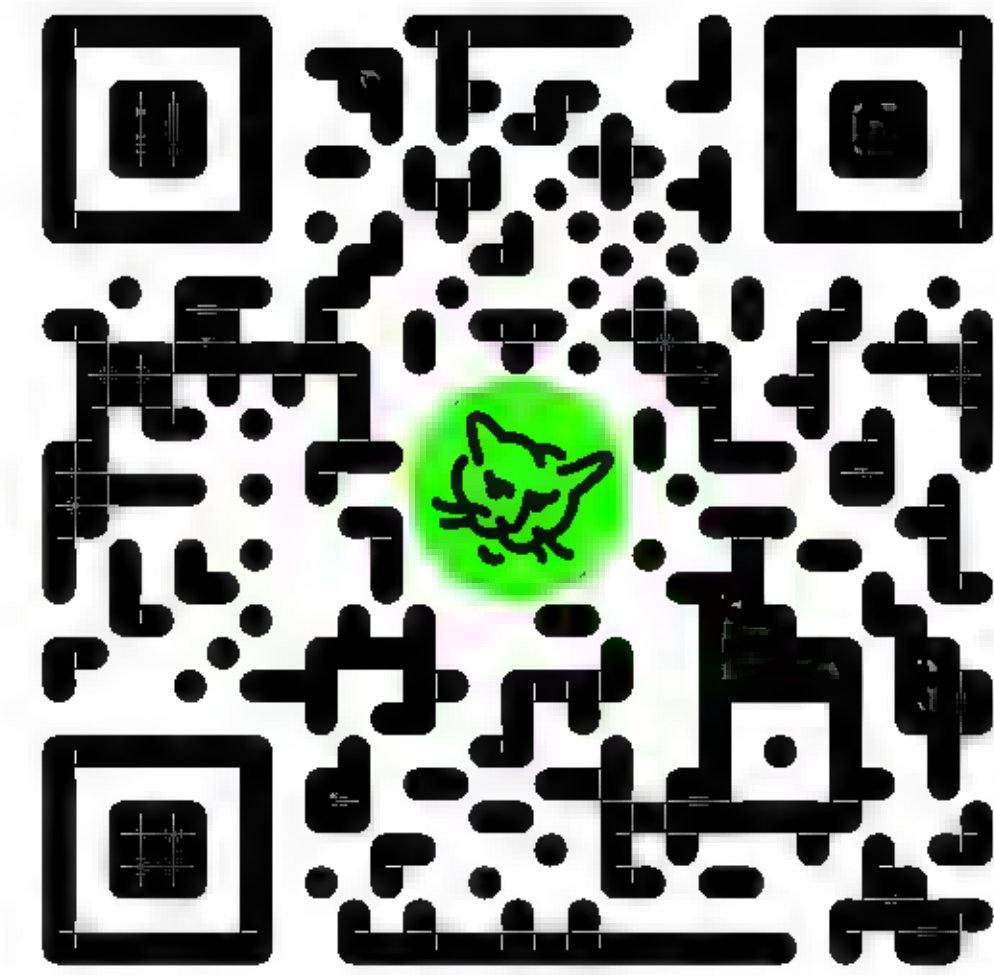
```
ELASTIC_HOSTS=http://localhost:9200 \  
ELASTIC_INDEX="filebeat-*" \  
LOKI_URL=http://localhost:3100 \  
python ./transfer.py
```

YOU CAN ALREADY USE IT

JUST ADD YOUR OWN DOCKER IMAGE, WE'LL DO THE REST:

```
$ helm repo add kts https://charts.kts.studio
$ helm repo update
$ helm upgrade --install \
  RELEASE_NAME \
  kts/es2loki \
  --set image.repository=your-docker-image
  --set image.tag=latest
```


SEE HOW IT WORKS



Components

1. Elasticsearch
2. Kibana
3. filebeat (imports "old" logs to Elasticsearch)
4. Grafana
5. Loki
6. Promtail (imports "new" logs to Loki)
7. PostgreSQL (needed for es2loki)
8. es2loki

Usage

In order to run a demo you may use:

```
docker compose up
```

[HTTPS://GITHUB.COM/KTSSTUDIO/ES2LOKI](https://github.com/KTSSTUDIO/ES2LOKI)

ANY PROFIT?



LOKI IN KUBERNETES

Pods(loki)[19]																
NAME↑	PF	READY	RESTARTS	STATUS	CPU	MEM	CPU/R:L	MEM/R:L	%CPU/R	%CPU/L	%MEM/R	%MEM/L	IP	NODE	QO	
loki-compactor-565b5b8565-wczqw	●	1/1	1	Running	16	58	30:0	100:600	53	n/a	58	9	10.233.88.135	wnode1	BU	
loki-distributor-64b94c49fc-8kpxl	●	1/1	0	Running	30	77	100:0	100:2048	30	n/a	77	3	10.233.88.30	wnode1	BU	
loki-distributor-64b94c49fc-ffq2z	●	1/1	2	Running	23	73	100:0	100:2048	23	n/a	73	3	10.233.73.103	wnode2	BU	
loki-distributor-64b94c49fc-q9ckb	●	1/1	0	Running	26	71	100:0	100:2048	26	n/a	71	3	10.233.126.26	wnode3	BU	
loki-gateway-55dcf67678-w8qww	●	1/1	0	Running	701	14	1000:0	50:500	70	n/a	28	2	10.233.73.134	wnode2	BU	
loki-index-gateway-0	●	1/1	2	Running	2	33	30:0	50:200	6	n/a	66	16	10.233.88.22	wnode1	BU	
loki-ingester-0	●	1/1	0	Running	32	899	100:0	3072:7168	32	n/a	29	12	10.233.88.207	wnode1	BU	
loki-ingester-1	●	1/1	0	Running	30	1233	100:0	3072:7168	30	n/a	40	17	10.233.88.185	wnode1	BU	
loki-ingester-2	●	1/1	48	Running	31	1459	100:0	3072:7168	31	n/a	47	20	10.233.88.216	wnode1	BU	
loki-memcached-frontend-0	●	2/2	0	Running	1	31	50:0	50:0	2	n/a	62	n/a	10.233.88.245	wnode1	BU	
loki-memcached-frontend-1	●	2/2	0	Running	2	31	50:0	50:0	4	n/a	63	n/a	10.233.126.24	wnode3	BU	
loki-memcached-frontend-2	●	2/2	1	Running	1	26	50:0	50:0	2	n/a	53	n/a	10.233.73.88	wnode2	BU	
loki-memcached-index-queries-0	●	2/2	0	Running	3	58	10:0	100:0	30	n/a	58	n/a	10.233.88.26	wnode1	BU	
loki-memcached-index-queries-1	●	2/2	0	Running	2	59	10:0	100:0	20	n/a	59	n/a	10.233.126.92	wnode3	BU	
loki-memcached-index-queries-2	●	2/2	2	Running	1	56	10:0	100:0	10	n/a	56	n/a	10.233.73.86	wnode2	BU	
loki-querier-866486df4d-2lhp7	●	1/1	0	Running	11	439	100:0	1024:5120	11	n/a	42	8	10.233.126.18	wnode3	BU	
loki-querier-866486df4d-lw94f	●	1/1	4	Running	8	453	100:0	1024:5120	8	n/a	44	8	10.233.73.101	wnode2	BU	
loki-querier-866486df4d-rmzw8	●	1/1	0	Running	12	441	100:0	1024:5120	12	n/a	43	8	10.233.88.58	wnode1	BU	
loki-query-frontend-7f89d8c6c7-5wn9j	●	1/1	0	Running	1	109	100:0	300:5120	1	n/a	36	2	10.233.126.219	wnode3	BU	

<namespace>

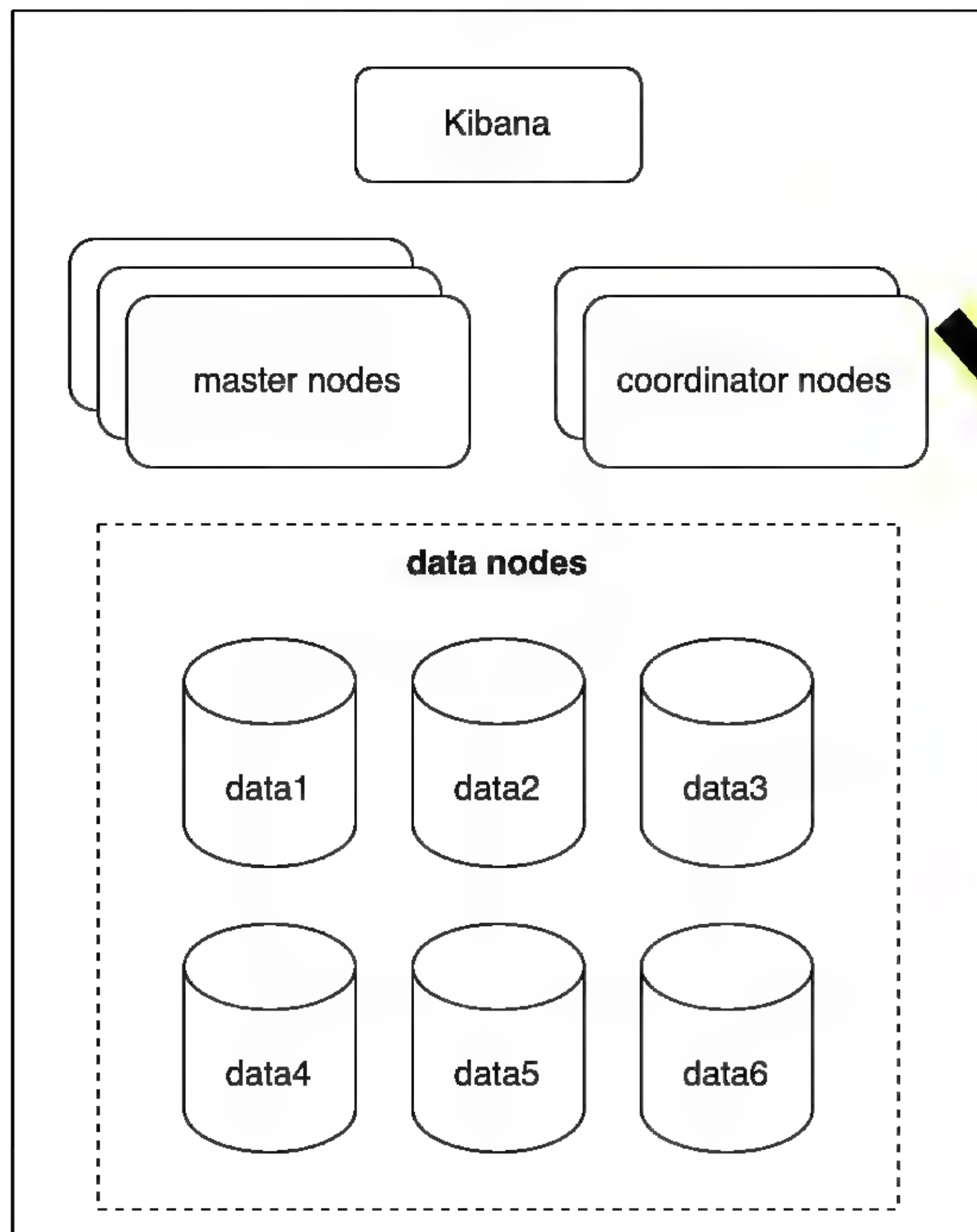
<pod>

11 * 3 cpu = 44 cpu

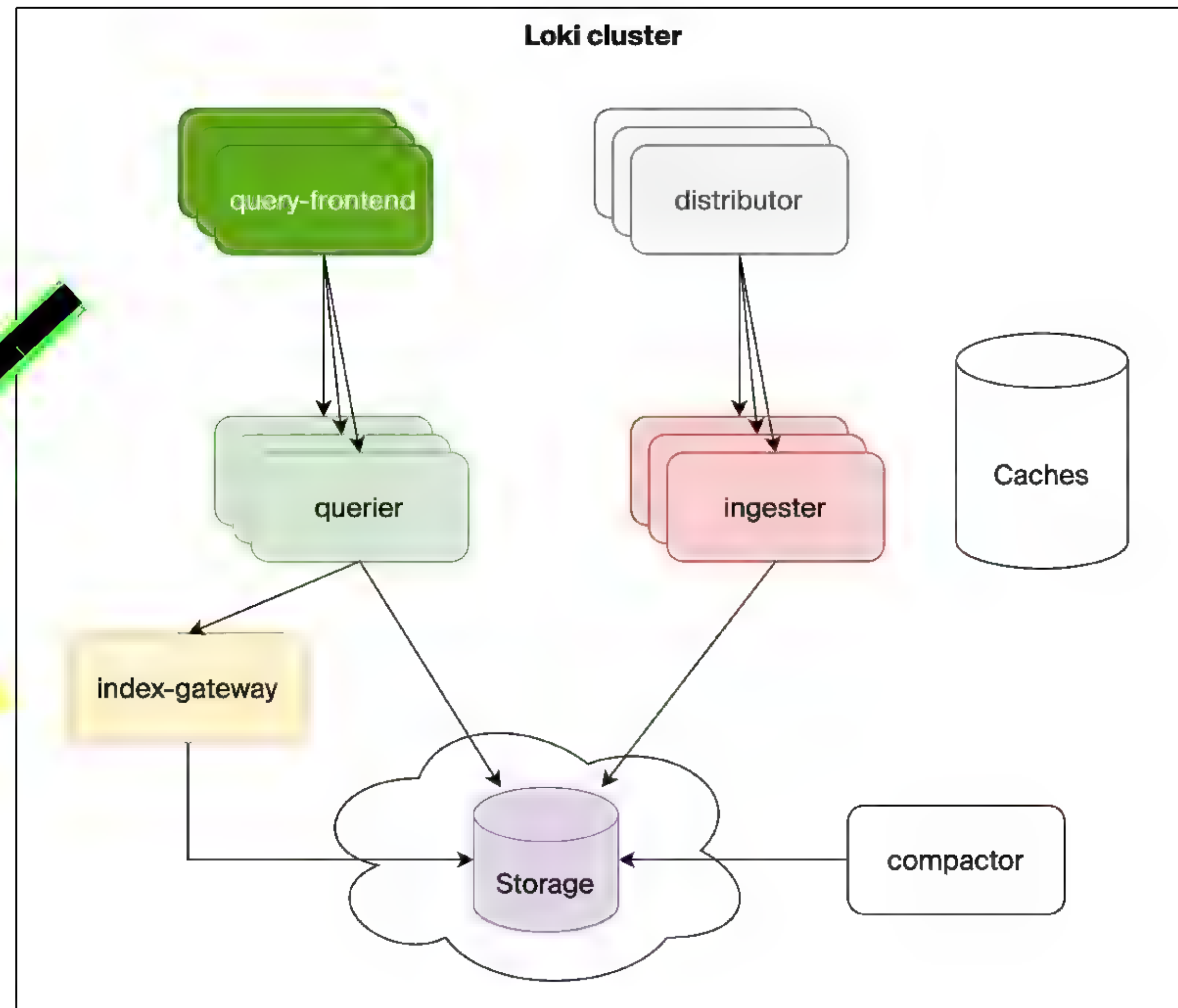
32 * 2 * 3 * 8 * 6 = 24 gb = 232 gb

ELASTICSEARCH VS LOKI

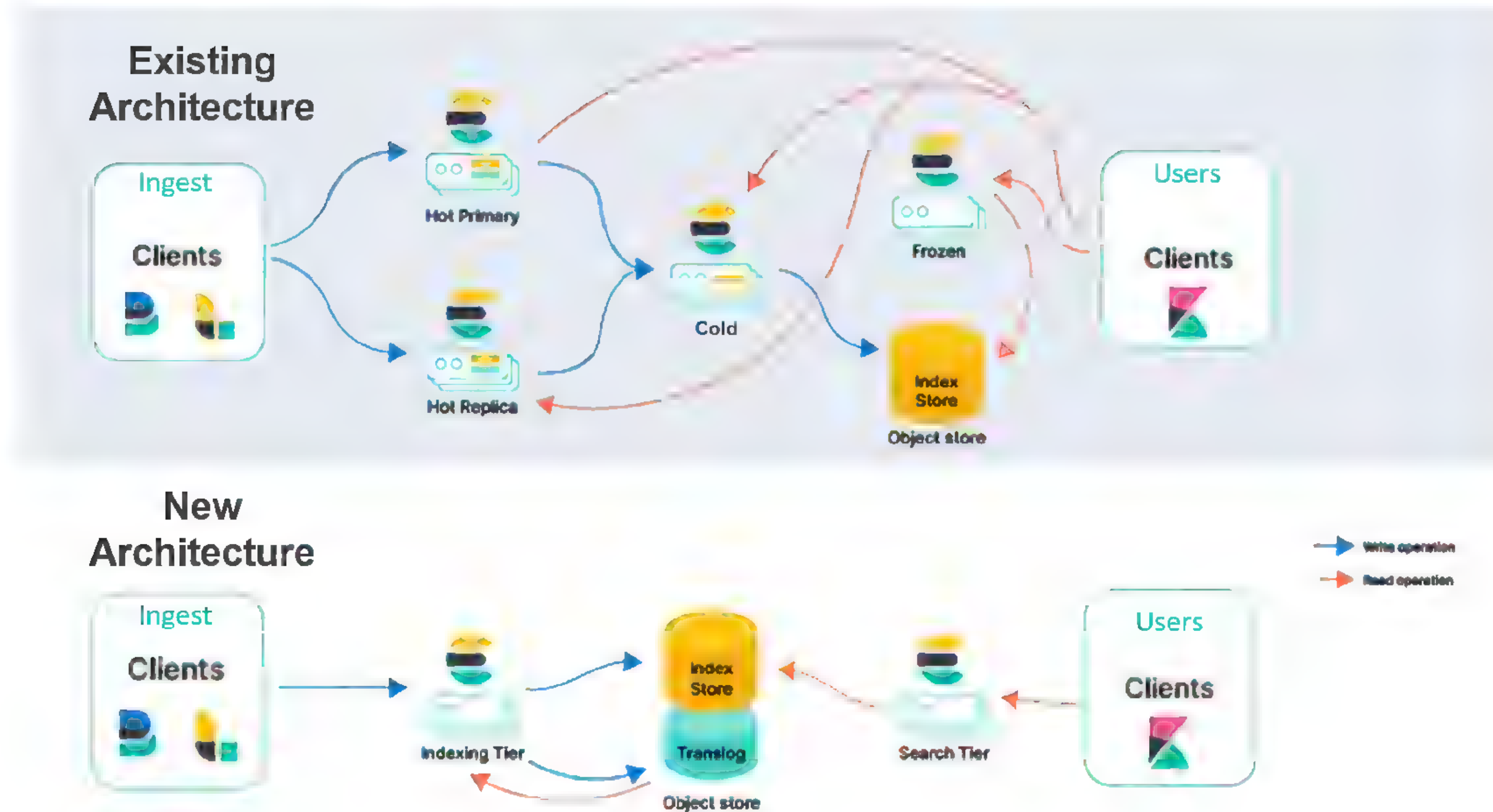
Elasticsearch cluster



Loki cluster



NEW STATELESS ELASTICSEARCH ARCHITECTURE



INFRA COMPARISON

ELASTICSEARCH

- CPU: 44
- Memory: 232 GB
- Main disks: 338 GB
- Storage disks: 25 TB



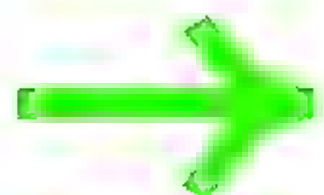
LOKI

- CPU: 7
- Memory: 14 GB
- Main disks: 105 GB
- S3 Storage: 5 TB

INFRA COMPARISON

ELASTICSEARCH

- CPU: 44
- Memory: 232 GB
- Main disks: 338 GB
- Storage disks: 25 TB



LOKI

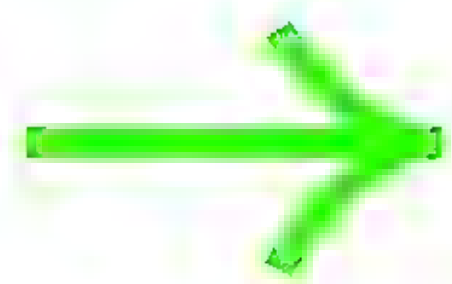
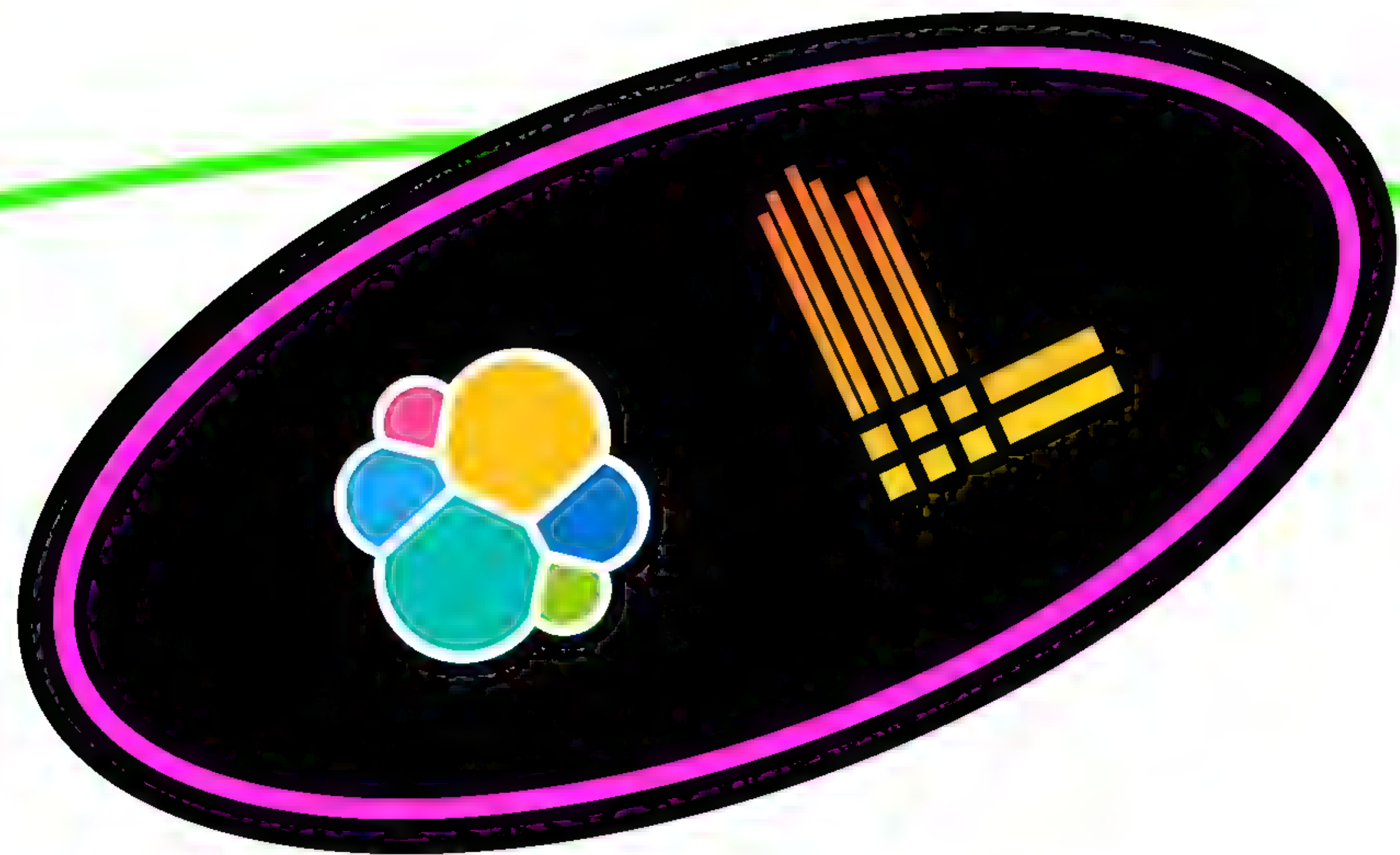
- CPU: 7
- Memory: 14 GB
- Main disks: 105 GB
- S3 Storage: 5 TB



SAVINGS

- 6x less CPU
- 16x less RAM
- 3x less disk
- 5x less main storage



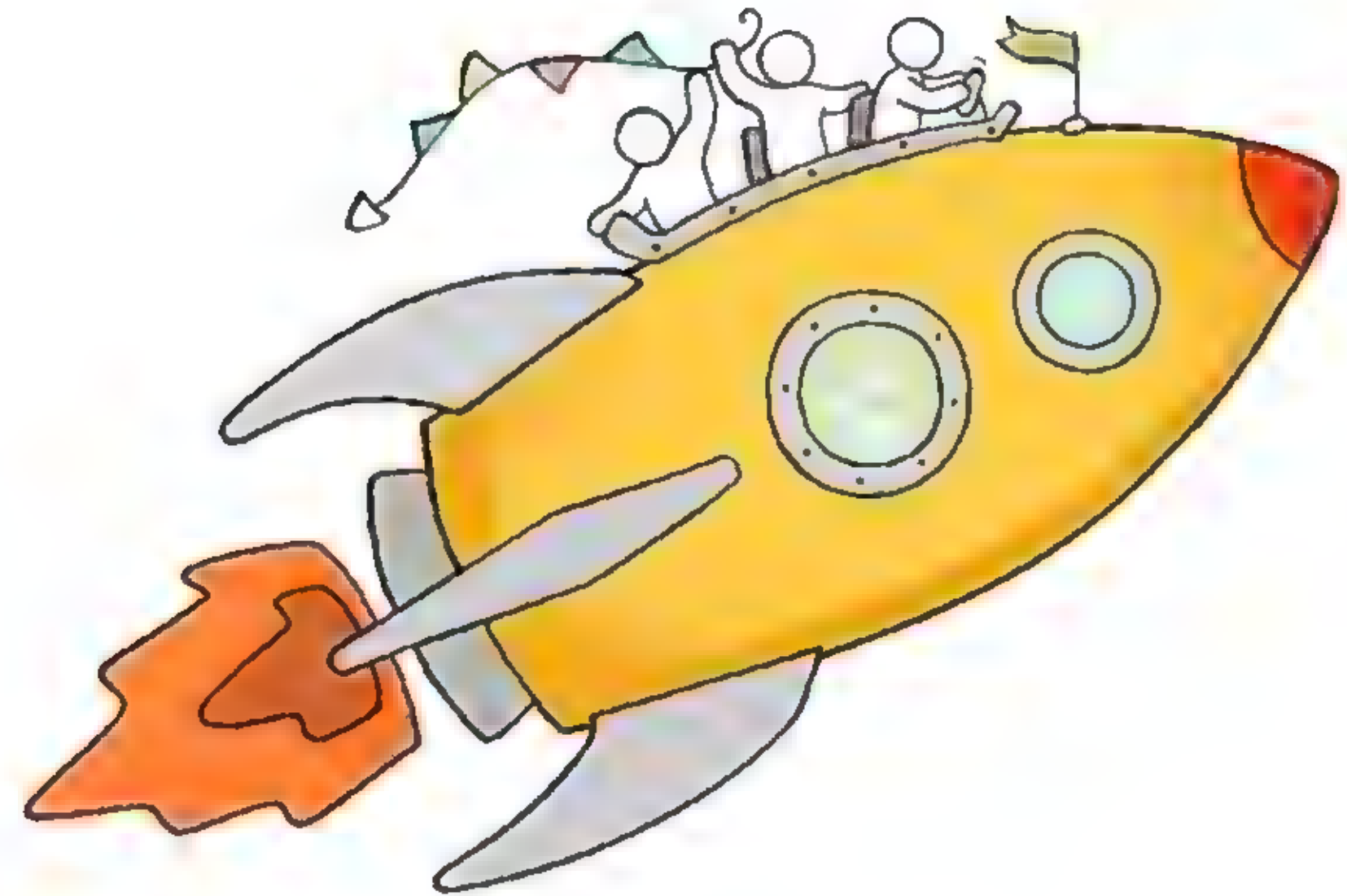


RECAP

RESULTS

POSITIVE :

- 1 4x logs infrastructure cost reduction
- 2 Simplier infrastructure maintanance
- 3 Much more stable installation than Elasticsearch
- 4 Logs transfer in adequate timespan
- 5 Easier to scale

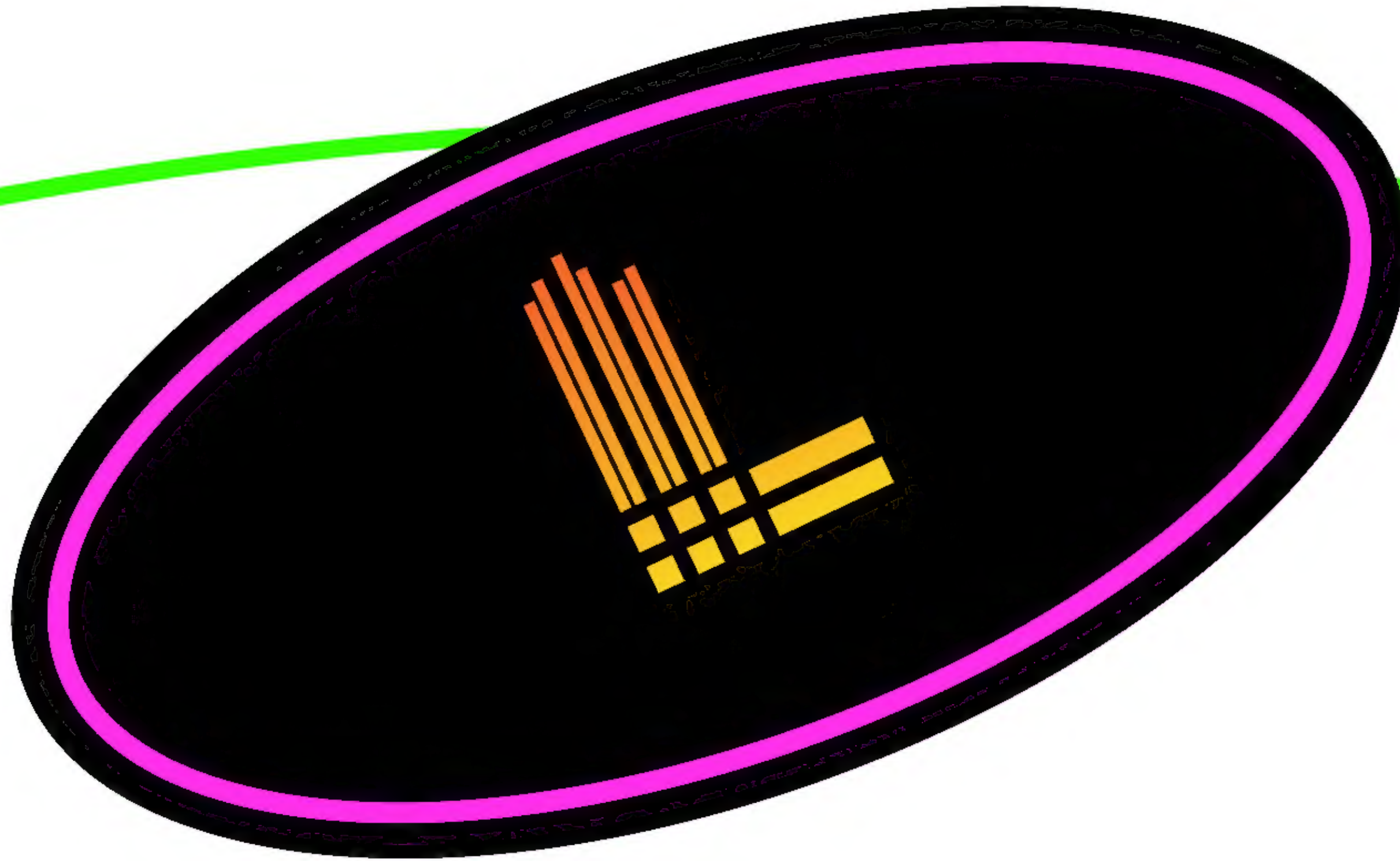


RESULTS

NOT SO POSITIVE, BUT OK:

- 1 Needed to use faster disks for S3
- 2 Grafana is not Kibana
- 3 No full-text search
- 4 No Machine Learning
- 5 Some queries in Loki are slower






LOKI CONFIGURATION
→ CHECKLIST (BONUS)

CONFIGURATION TIPS

Essential Grafana Loki configuration settings

Share this: 



t.ly/vMvK

TECH TALKS - DECEMBER, 16

DYNAMIC
ENVIRONMENTS
WITH ARGOCD AND
CROSSPLANE

→ 15:30 - 15:50



MLOPS
IN ENTERPRISES
AND SMB

→ 15:30 - 15:50



Leave your feedback!

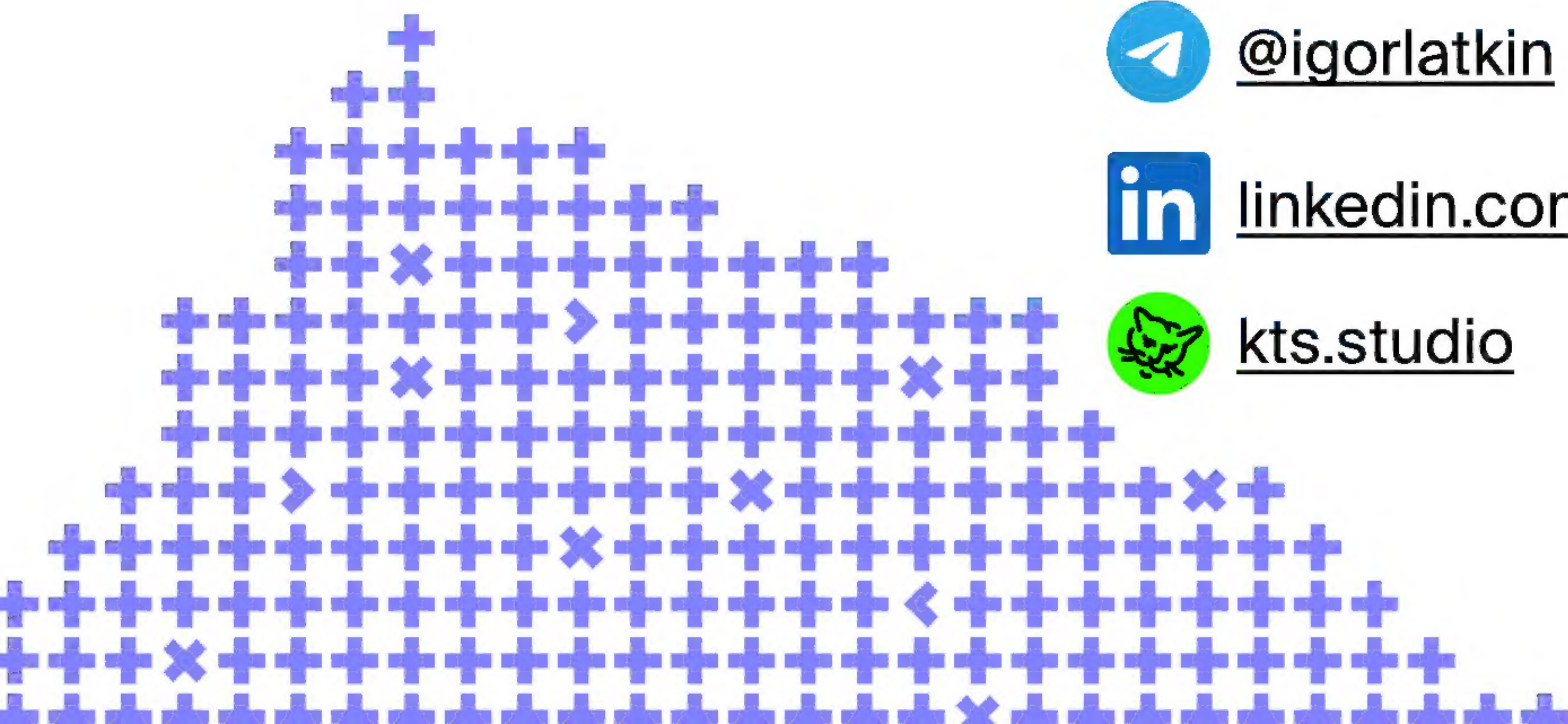
**You can rate the talk
and give a feedback on
what you've liked or
what could be
improved**



 [@igorlatkin](https://t.me/@igorlatkin)

 [linkedin.com/in/igorlatkin](https://www.linkedin.com/in/igorlatkin)

 kts.studio



Co-organizer

Yandex